

**AFRL-IF-RS-TR-2002-141**  
**Final Technical Report**  
**June 2002**



# **COMPUTATIONAL IMMUNOLOGY FOR THE DEFENSE OF LARGE SCALE SYSTEMS**

**Odyssey Research Associates, Incorporated**

**Sponsored by**  
**Defense Advanced Research Projects Agency**  
**DARPA Order No. F163**

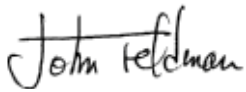
***APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.***

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

**AIR FORCE RESEARCH LABORATORY**  
**INFORMATION DIRECTORATE**  
**ROME RESEARCH SITE**  
**ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-141 has been reviewed and is approved for publication.

APPROVED: 

JOHN FELDMAN  
Project Engineer

FOR THE DIRECTOR: 

WARREN H. DEBANY, Technical Advisor  
Information Grid Division  
Information Directorate

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 2002	3. REPORT TYPE AND DATES COVERED Final Aug 97 – Aug 99		
4. TITLE AND SUBTITLE COMPUTATIONAL IMMUNOLOGY FOR THE DEFENSE OF LARGE SCALE SYSTEMS		5. FUNDING NUMBERS C - F30602-97-C-0216 PE - 62301E PR - F163 TA - 40 WU - 04		
6. AUTHOR(S) Carla Marceau, Matthew Stillerman, Maureen Stillman, Stephanie Forrest				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Odyssey Research Associates, Incorporated Cornell Business & Technology Park 33 Thornwood Drive, Suite 500 Ithaca New York 14850		8. PERFORMING ORGANIZATION REPORT NUMBER  N/A		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency AFRL/IFGB 3701 North Fairfax Drive 525 Brooks Road Arlington Virginia 22203-1714 Rome New York 13441-4505		10. SPONSORING / MONITORING AGENCY REPORT NUMBER  AFRL-IF-RS-TR-2002-141		
11. SUPPLEMENTARY NOTES AFRL Project Engineer: John Feldman/IFGB/(315) 330-2664/ John.Feldman@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 Words)  This report describes the application of the computational immunology approach to a distributed object systems. The hypothesis tested was that one could characterize normal behavior of the application itself in terms of inter-object messages, and use that characterization to successfully detect rogue client attacks on the application. The goals of the research were to test and demonstrate the feasibility of intrusion detection at the application level in distributed object systems. In particular, we worked with applications built on the Common Object Resource Broker Architecture (CORBA). The report shows that the computational immunology approach reliably detects attacks on the Domain Name Server that seriously disrupt Internet service.  The report analyzes the components required for a definition of "self" that is applicable to computer programs. The report also conducts experiments that show that a straightforward definition of "self" can detect rogue client attacks on CORBA systems. The project resulted in building a prototype system to aid in the analysis of experimental data and helped generate descriptions of normal application behavior. The prototype intrusion detection system for CORBA can be used with a broad class of definitions of "self".				
14. SUBJECT TERMS Intrusion Detection, Computational Immunology, Anomaly Detection, Distributed Systems, Corba Intrusion Detection, CIDF Standard.			15. NUMBER OF PAGES 69	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT  UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE  UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT  UNCLASSIFIED	20. LIMITATION OF ABSTRACT  UL	

# Table of Contents

Table of Contents .....	i
Figures .....	iii
Tables .....	iv
1 Introduction .....	1
2 Lessons Learned .....	3
2.1 The Threat of a Rogue Client Attack .....	3
2.2 Computational Immunology Detects Rogue Client Attacks .....	4
2.3 Local vs. Statistical Anomaly Detection .....	4
2.4 Defining “Self” in Computational Immunology .....	4
2.5 Issues In Empirically Creating A Self Database .....	5
2.6 Efficient Implementation Of A Self Database .....	6
2.7 Negative vs. Positive Detection .....	6
3 Research at ORA: Applying Computational Immunology to Distributed Object Applications .....	6
3.1 A Definition of Self For CORBA Applications .....	7
3.2 Experimental Design .....	11
3.3 Results .....	13
3.3.1 The PersonnelTracker experiment .....	13
3.3.2 The LPA Vision experiment .....	19
3.3.3 Detection efficiency and false-positive rate .....	22
3.4 Discussion .....	23
4 Research at the University of New Mexico .....	24
4.1 Data Modeling .....	24
4.2 Real-Time Monitoring and Response .....	27
4.3 Papers Published .....	28
5 Software Design and Development .....	28
5.1 The Design of the CORBA Immune System .....	28
5.1.1 What the system does .....	29
5.1.2 Overview of how the system works .....	29
5.1.3 Implementation overview .....	30
5.1.4 Finite-state machine implementation of the self database .....	32
5.2 The Design of the IDA .....	35

6 CIDF Promotion to the Government and Commercial Sectors.....	38
6.1 Summary of Vendor Visits and Accomplishments.....	38
6.1.1 HP (October 5, 1998) .....	39
6.1.2 Centrax (August 5, 1998).....	39
6.1.3 Network Associates (October 5, 1998) .....	40
6.1.4 Newbridge (November 17, 1998).....	40
6.1.5 Axent Technologies (November 16, 1998) .....	40
6.1.6 CISCO (August, 1998).....	40
6.1.7 ISS .....	40
6.1.8 SAIC .....	40
6.2 Vendor Questionnaire .....	41
6.3 Lessons Learned .....	42
7 Conclusion .....	43
8 References .....	45
Appendix.....	47
Questionnaire Responses from Intrusion Detection Vendors.....	47
Axent Technologies.....	47
Centrax .....	49
HP .....	52
Network Associates.....	56
SAIC.....	59

## Figures

Figure 1. Coverage for $N = 1$ .	15
Figure 2. Coverage for $N = 2$ .	15
Figure 3. Coverage for $N = 3$ .	16
Figure 4. Coverage for $N = 6$ .	16
Figure 5. Anomaly graph for 55.37, $N = 2$ .	18
Figure 6. Anomaly graph for 51.24, $N = 2$ .	18
Figure 7. Anomaly graph for 51.24, $N = 3$ .	19
Figure 8. Coverage graph for $N = 1$ .	20
Figure 9. Coverage graph for $N = 2$ .	21
Figure 10. Coverage graph for $N = 3$ .	21
Figure 11. Anomaly graph for normal traces.	23
Figure 12. Overview of the CORBA Immune System.	30
Figure 13. A partial tree for the example string.	33
Figure 14. Self-detector generated by the IDA.	35
Figure 15. Self-detector and the R/A Module.	36
Figure 16. User interface of the IDA.	37

## Tables

Table 1. Size of self database as a function of window width.....	14
Table 2. Summary of PersonnelTracker experimental results.....	17
Table 3. Results of the LPA Vision experiment. ....	22
Table 4. Company and commercial product matrix.....	39
Table 5. Requirements for intrusion detection standards.....	41
Table 6. Openness of product and security services.....	42

## 1 Introduction

This report describes the collaborative research in the area of intrusion detection performed by Odyssey Research Associates in collaboration with Stephanie Forrest at the University of New Mexico.

Our approach to intrusion detection is based on Forrest's pioneering work in computational immunology, that is, the design of computer defense systems inspired by the vertebrate immune system. (An introduction to computational immunology can be found in [4].) Although vertebrate immune systems offer many promising features, the research in this project focuses on the following three properties: *anomaly-based* intrusion-detection systems that characterize programs by their behavior, which is determined empirically. We adopt this approach because it is unrealistic to require the application developer either to include checks for intrusions in the application code or to write specifications for how he expects the code to be used. In our opinion, approaches that require extra work from the application developer—work that is not directly concerned with application functionality—are bound to fail.

In the current project, we are particularly interested in applying the computational immunology approach to *distributed object systems*, a relatively new and increasingly popular technology for building applications. Intrusion detection at the application level can complement traditional intrusion detection and other computer defenses such as access controls that protect host machines and networks. In particular, application-level intrusion detection may have a special role in fighting insider attacks, or misuse of the application; such attacks are more common than external attacks and pose a significant threat to our military and industrial infrastructure. In this report, the term “intrusion detection” also includes “intrusions” in the form of insider attacks.

We are interested particularly in two kinds of attack on distributed object systems. These are the *rogue client attack* and *misuse* attacks. Modern distributed object systems, with well-documented interfaces, give attackers access to the “insides” of an application. For example, a complex business application might consist of a front-end user interface, which establishes the user's identity and determines what functionality he is allowed to exercise, and a back-end, which is a database server. The correct functioning of the application depends on the back end being able to assume that requests from the front end are legitimate. Such an assumption is unwarranted if the user can write his own front end, which we call a *rogue client*, and use that instead of the authorized one. One might imagine that it is too difficult to



write a new front end, but that is not the case. A rogue client can easily be generated either by modifying the normal front end or by using a simple testing tool that exercises the back end.

Our hypothesis was that we could characterize normal behavior of the application itself in terms of interobject messages, and use that characterization to successfully detect rogue client attacks on the application. In the example case, an “immune system” would be able to distinguish between the legitimate front end and unauthorized ones, and could reject requests from the latter.

We further hypothesized that the same mechanism could be helpful in detecting *misuse attacks*, which occurs when an authorized user of the system has legitimate access but uses it in ways that exceed his authority. For example, consider a parts-planning application. A parts planner may have access to the central parts-planning database, but is expected only to review and manage those parts entrusted to him. Entering spurious orders for parts managed by a co-worker is not authorized.

The goals of our research project have been, in brief,

- To test/demonstrate the feasibility of intrusion detection at the application level in distributed object systems. In particular, we worked with applications built on the Common Object Resource Broker Architecture (CORBA) [12] as a typical platform for building distributed object applications.
- To verify that immune-style anomaly detection is a reasonable way to detect intrusions at the application level.

In this report, we describe the work that we did and our results to date. Our major results are described in Section 1. Briefly, we

- showed that the computational immunology approach reliably detects attacks on the Domain Name Server that seriously disrupt Internet service;
- analyzed the components required for a definition of self that is applicable to computer programs and reported the analysis in *Communications of the ACM* [15];
- conducted experiments that show that a straightforward definition of self can detect rogue client attacks on CORBA systems;
- built a prototype system to aid in the analysis of experimental data and generate descriptions of normal application behavior;
- built a prototype intrusion detection system for CORBA that can be used with a broad class of definitions of self; and
- participated in a successful demonstration of intrusion detection systems communicating by means of the Common Intrusion Detection Framework (CIDF) protocol [14].

Additionally, in cooperation with other researchers in the area of intrusion detection, we worked on the CIDF definition and promoted the concept of standards for cooperation among intrusion detection systems.

The remainder of this report is organized as follows. Section 2 summarizes the lessons we learned from this research effort. Section 3 describes our ongoing experiments on the definition of self at Odyssey Research Associates; Section 4, the work at the University of New Mexico. Section 5 describes the design of the CORBA Immune System, our system for intrusion detection for CORBA applications. Section 6 describes our work on the CIDF standard. An appendix contains the results of a survey of intrusion detection vendors, undertaken as part of the CIDF promotion effort.

## **2 Lessons Learned**

We applied intrusion detection techniques to the problem of protecting distributed applications. Our experiments showed that the computational immunology approach can effectively protect a large class of such applications against certain important kinds of attacks. By implementing a system based on computational immunology, we also learned about the obstacles that remain before practical immunology systems are feasible.

### **2.1 The Threat of a Rogue Client Attack**

Distributed object systems with published interfaces support the development of open, modular applications. However, some applications are designed with close cooperation between their parts. The application designer often finds it advantageous to create one module that maintains certain invariants, on which other modules then rely. (In fact, this kind of design is commonplace.) For example, module *M* may maintain the invariant that the value of debits submitted equals the value of credits. A program masquerading as *M* can compromise the entire application by submitting unbalanced credits or debits. We call such a module a *rogue client*.

Security features for distributed applications typically include encryption and (user and/or host) authentication, but do not ensure that the client sending a message is actually executing the expected code. For example, such assurance is not part of the CORBASEC security standard for CORBA [13]. Behavioral-based intrusion detection, such as the CORBA Immune System, can provide such supplementary assurance.

## 2.2 Computational Immunology Detects Rogue Client Attacks

Our experiments showed that the computational immunology approach (detect *anomalies* by comparing actual *behavior* against an *empirically* derived norm) detects rogue clients. We performed the experiments using the Orbix implementation of CORBA, but the results apply to any distributed object-oriented platform.

The CORBA Immune System detects anomalies based solely on the request method; parameters to the requests are ignored. Because it detects anomalies based on the sequence of requests from client to server, it is effective with applications that support a sufficiently rich set of methods. We encountered one (military) application that supported a single method (which might be called “do\_all\_work”); all information about the work to be done was encoded in the parameters to that operation. Our method could be extended to include parameters to the requests. Whether it is worth doing so depends on the number of applications that employ a similar encoding.

## 2.3 Local vs. Statistical Anomaly Detection

Our first attempts at anomaly detection measured the percentage of anomalous events in the entire sequence of requests from client to (server) object. We abandoned this attempt, and we are convinced that it is unworkable. Any attempt to characterize a client based on the entire sequence of its requests to a given server is doomed to fail. It may well be that only a small portion of a rogue client is anomalous. For example, it would be a simple matter for an insider to modify a legitimate client program so that most of the program would behave in a normal way. Only a small part would be anomalous. By running the program normally for a long time, an attacker could make the percentage of abnormality arbitrarily small.

## 2.4 Defining “Self” in Computational Immunology

Computational immunology is based on a definition of self, or “normal.” We have found that any such definition has four components: focus, data stream, projection, and detector algorithm. *Focus* describes what entity can be either normal or anomalous. In our case, the focus is the client that sends requests to a distributed object. *Data stream* is the source of data used to differentiate between normal and anomalous (or self and non-self). *Abstraction* describes how the data stream can be divided into equivalence classes that discriminate between normal and anomalous. The *detector algorithm* describes how to decide when an entity is sufficiently anomalous that it can be considered an attack.

Most research efforts in computational immunology to date have focused on the last two of these four components. However, the first two are the most

important. Once we had identified the data stream of client/server requests, no great ingenuity was required to find a projection and a detection algorithm. In our CORBA system, we experimented with the sliding window algorithm, varying the width  $N$  of the window. Since CORBA requests are fairly coarse-grained, we could detect anomalies for all values of  $N$  greater than 1. Values larger than 3 simply made it harder to collect enough training data to cover all normal behavior. Given the right data stream, surprisingly weak algorithms are sufficient [16].

## 2.5 Issues In Empirically Creating A Self Database

The great challenge in developing an empirical self database is creating the description of “normal,” which we call the self database. The self database problem has two parts: ensuring that the training data covers all normal behavior, and maintaining the self database.

First is the difficulty of collecting enough training data to cover all normal behavior—we call this obtaining *coverage*. Normal data can be generated either synthetically (by a program or script) or empirically by users going about their normal activity. Each method has its inherent difficulties. *Synthetic normal* is a guessing game as to what is normal. *Empirical normal* also has its risks. Suppose training is conducted during a time when work is focused on a relatively small subset of legitimate activity. The result is reminiscent of the joke about the man with 20 years experience; unfortunately, it was the same year of experience repeated 20 times. The test users must exercise the system doing a broad spectrum of real tasks. However, with both methods, there is always the question, “What normal activity has been left out?” Empirical normal also poses the risk that an attack might occur during training—it is difficult to ensure that this will not happen.

The second problem is that both methods are labor intensive. Gathering empirical data and evaluating whether the normal has been obtained requires significant input from a highly trained person. More research is needed on how to automate the generation and maintenance of the self database. This problem is exacerbated by the fact that the self database needs to be recomputed with every release and every installation (the actual database content depends in general on installation parameters).

One possible solution to the problem of maintaining the self database is incrementally aging and modifying it as new releases appear and standard usage behaviors change. It is, of course, desirable to do this without having to recompute the entire database.

## **2.6 Efficient Implementation Of A Self Database**

The sliding window algorithm for program behavior combines conceptual simplicity with effectiveness. Recent work at the University of New Mexico has shown that more complicated algorithms are not needed [16]. On the other hand, researchers have experimented with even simpler algorithms in the search for increased efficiency. The risk is that such simple algorithms may increase the likelihood of false negatives, much as very small values of  $N$  do.

We show in Section 5.1.4 that the sliding window database can be constructed in the form of a finite state machine (FSM). Given the next item in the data stream, the FSM declares the window ending at that element normal or anomalous; it then proceeds to the next state. This provides a very efficient implementation of the sliding window algorithm.

Previously, it was thought that an FSM implementation was infeasible, because of the explosion of states that seemed to be required to recover from inputs that are not in the self database (e.g., see [6]).

Furthermore, the FSM can be constructed on the fly as training data is entered, thus making feasible an automatic switch between training mode (building the self database) and detection mode (testing data against the self database). This on-the-fly training should enable the system to adapt to new behaviors.

## **2.7 Negative vs. Positive Detection**

Our approach catches rogue attacks, but it does not give any further information about the attack. All that the detector “knows” about the rogue client is that it is not the normal client. The vertebrate immune system is smarter. It would be possible for the CORBA Immune System (or any anomaly-based intrusion detection system) to capture the identifying markers for the rogue client and store them in a “rogues’ gallery,” for future identification. Rogues’ gallery information could be shared with other servers, much as virus information is today, and some form of positive detection could be used for known rogues.

## **3 Research at ORA: Applying Computational Immunology to Distributed Object Applications**

The central idea of “computational immunology” is anomaly detection based on a concept of “self” and distinguishing self from other. We spent some time examining the notion of a definition of self in a computational environment. “Self” has been defined for a variety of different entities. What

constitutes a definition of self in general, and for distributed applications in particular? This work, summarized in Section 3.1, was published in *Communications of the ACM* [15].

Testing our definition of self was not a trivial task. Networks and operating systems “enjoy” an established body of known exploits, against which an intrusion detection system can be tested. There is no such collection of attacks on distributed applications. Therefore, we had to find and/or create an application, as well as attacks against it. This proved to be easier for rogue client attacks than for misuse attacks.

We experimented with a large commercial application and with a small application that we wrote ourselves. The experiments that we performed are described in Section 3.2. The results are in Section 3.3. We discuss them further in Section 3.4.

### **3.1 A Definition of Self For CORBA Applications**

To define “self” (or normal behavior) for distributed object applications, we first formulated essential characteristics of Stephanie Forrest’s definition of self for Unix processes and then applied that understanding to CORBA applications.

There are several examples of definitions of self. Vertebrate immune systems are (of course) examples, and Stephanie Forrest’s work on a definition of self for Unix processes is well known. Other work by Forrest has produced successful definitions of self in quite different areas, such as detection of computer viruses and novelty detection in time-series data. In all of these cases, we generally observe some empirically-formed characterization of normal, together with an algorithm for comparing observations “in the field” with that characterization.

We abstracted the concept of a definition of self in order to isolate its components. A paper describing this work appeared in the July 1999, issue of *Communications of the ACM* [15]. To use the immunological technique, details of how that characterization is formed and how that comparison is made must be settled. These choices are what we call the “definition of self.” The following discussion assumes some familiarity with Forrest’s work on computer immunology, e.g. [4].

Here we briefly describe the components of a definition of self and use as an example our current definition of self for CORBA applications, a definition that is analogous to Forrest’s definition of self for Unix processes. The four components are focus, data stream, abstraction, and detection algorithm.

**Focus.** The first component of a definition of self is the entity whose “self” we are describing. Using biological terms, we need to define the organism to

be defended—in this case, a CORBA application—and the “cell”—that is, that entity within the application that we will examine to ascertain if it is normal. CORBA applications use a client/server model of communication: clients are programs that invoke the operations of objects, while servers are processes that own and manage objects. An application might have one or more servers. Each server manages objects of a specific type or types. As discussed earlier, we decided to focus on application clients as seen from the vantage point of application servers. We want to instrument those application servers that manage important resources, so that we can detect malicious client actions. In particular, we focus on each client/server pair, which is called a *connection*.

**Data collection.** The second component of a definition of self is defining what observable aspect of the “cell” we can use to distinguish normal from abnormal behavior. We chose to look at the message traffic between clients and servers within the application. CORBA defines the conventions of that traffic and ways for us to monitor it, so it is possible to construct a generally applicable system based on that communication. In addition, the traffic is essential to getting things done, and so should tend to expose misuse of the application.

We used the Orbix™ Object Request Broker (ORB) [9] for our experiments. Orbix provides a general facility for intercepting messages called *filters*. A filter is a C++ class that the user subclasses and instantiates within a server or client program. Orbix interposes the instantiated filter in the message path, where it can observe and act on each message. Our experimental setup *records* information about the requests in a file. Many other ORBs provide a facility similar to filters. *Interceptors* are a standard for such facilities that has recently been adopted by the OMG. Note that filters (and eventually interceptors) are also used by our CORBA Immune System prototype to detect messages as part of an operational intrusion detection system (see Section 5.1.3 and [11]).

It is essential to be able to instrument the application in a non-intrusive way so as not to require a lot of work on the part of application developers, who have scant free time to devote to intrusion detection. As discussed earlier, CORBA in general and Orbix in particular provide convenient ways to monitor message traffic between client and server. Thus, for example, in the experiments described later (see Section 3.2), we were able to collect traffic data for a commercial application at the developer’s office without interfering with the developers’ work.

**Abstraction.** Messages are extremely diverse, and thus not readily comparable. A definition of self includes a way to define equivalence classes on the message data that are useful in discriminating between normal and

abnormal usage. The equivalence class of a sequence of messages is discernable from (and defined by) its *signature*, an abstraction (or projection) of the message data. By extension, we speak of the “signature of a connection.”

The abstraction we are currently investigating depends *only* on the sequence of requested methods from the client to the server, and ignores such details as the arguments, the time of the request, and so on. Furthermore, this abstraction only preserves *local* sequentiality: The signature of the connection is the *set* of all substrings of length  $N$  of this sequence, as obtained by a sliding window algorithm (as described in [5]). Following Forrest, we define the self database as the union of all signatures (i.e., of those sets) encountered during normal operation.

In practice, as with all anomaly-based intrusion detection systems, we posit a training period, during which we collect data from presumably normal connections. Their signatures comprise the self database.

Note that although our definition of self is closely modeled on Forrest’s definition for Unix processes, there is at least one important difference. Unix kernel calls are relatively fine-grained. CORBA operations (methods on distributed objects), by contrast, tend to be coarser-grained. This is because in a distributed system, developers know that a request to an object may involve interhost communication, which is much more expensive than a Unix kernel call. One way to look at our research is that we are studying how to use shorter sequences of more powerful requests for computer immunology.

An important variable in the sliding window algorithm is the size of the window, since this has a dramatic effect on both the size of the self database and the effectiveness of anomaly detection. See Section 3.3.1.1 for a discussion of the effect of window size in our experiments.

**Detection algorithm.** The first stage of detection is to see if each of the substrings obtained at runtime from a connection is in the self database. The result is a sequence<sup>1</sup> of yes/no values. A fast finite state machine algorithm for computing this Boolean sequence is described as part of the architecture of the CORBA Immune System (see Section 5.1.4).

It is necessary to obtain from the yes/no sequence some overall *anomaly measure* for the connection that at any one time can tell us how closely the connection as a whole conforms to the self database. Attacks tend to be

---

<sup>1</sup> The sequence preserves the order in which these substrings occur on the connection, as the events are viewed through a sliding window.



characterized by many anomalies bunched together. We used Forrest's *locality frame measure* so that our measure would be sensitive to bunching. The locality frame measure is computed as follows. Consider a sequence of events,  $e_1, e_2, e_3, \dots$ . Events  $e_1$  through  $e_N$  define the first window of values. If that sequence is *not* in the self database, then the sequence is anomalous. Sliding the window, we consider  $e_2, \dots, e_{(N+1)}$ . Again, it may be anomalous or not. Looking at the first  $L$  anomaly values (where  $L$  is the locality frame size), we can count how many of them are anomalous—this number is the locality frame measure. A high locality frame measure indicates that a large number of anomalies occur together, making an attack more probable. In this section, we use a locality frame size of 20.

In practice, even after extensive training, we can expect a certain number of mismatches between connections encountered in operation and the self database that we obtain from training. The question is, how bad does the mismatch have to be before we suspect an intrusion? We postulate a *threshold* value for the anomaly measure that can distinguish between normal variation and the degree of anomaly that characterizes an intrusion. When attack connections have clearly different anomaly values from normal connections, it should be possible to define good distinguishing threshold values.

In summary, our candidate definition of self for CORBA applications can be characterized by the following choices in the above four categories.

- We *focus* on the clients of certain important servers, which we call the instrumented servers.
- We *collect data* about requests from clients to the instrumented servers.
- We examine in particular the method being requested. The sequence of requested methods between each client/server pair is scanned through a sliding window to extract all fixed-length substrings. The sequence of substrings is our *abstraction* (signature) of the client/server pair.
- Our *detection algorithm* is as follows. Each of these substrings is compared with a database of substrings found during a training period, during which we assume that no intrusions are occurring. If the substring exactly matches an entry in the database, it is considered *normal* and its anomaly value equals *zero*. Otherwise, the substring is considered anomalous, and its anomaly value is 1. The sequence of anomaly values for each client/server pair is aggregated continuously into a running anomaly measure, sensitive to both the number of anomalies seen and how recently (in the sequence) they occurred. We are ultimately concerned with local values of the anomaly measure for each connection, and whether those values exceed a threshold.

### 3.2 Experimental Design

In this section, we discuss the experiments we performed to test our candidate definition of self for CORBA applications.

For any intrusion detection system, two primary concerns are *detection efficiency* and *false alarm rate*: Given that an intrusion occurs, what is the likelihood that it will be detected? What fraction of IDS alarms corresponds to real intrusions? Our experiments, therefore, are of two types:

- Running the IDS algorithm with data that definitely contains an intrusion to see if it is detected.
- Running the IDS algorithm with data that does not contain any intrusions to see if false alarms are raised.

The system operates in two phases: During the *training* phase, message traffic is collected and characterized concisely. This characterization serves as a standard of normal behavior during the *detection* phase, when similar data, possibly reflecting an intrusion, is collected.

Our IDS is intended to work with CORBA applications—to detect when those applications have been compromised or are being abused. We therefore tested the IDS algorithm with two CORBA applications, a large commercial application called *LPA Vision* [10] and a small application developed at ORA called *PersonnelTracker*. Both applications are hosted on the Orbix ORB.<sup>2</sup>

LPA Vision, developed by LPA Software of Rochester, New York, is widely used by parts planners to predict and control the inventory of parts for a company. Typical activities for a planner include

- extrapolating demand and supply of parts into the future,
- ordering more parts, or adjusting existing orders, to correct anticipated imbalances,
- reconciling differences between projected and actual demand, considering alternative forecasting methods, and
- generating reports on the current state of the parts inventory and planning system.

Working with LPA Software, we instrumented the LPA Vision servers and clients, as described earlier. In several sessions, the software was then

---

<sup>2</sup> Attempts to obtain a suitable military application did not succeed. Two applications that were available to us proved to be unsuitable because the request vocabulary of the servers was very small. In one case, there was a single request, on the order of “do all work.” All pertinent information was in the arguments to the requests—unfortunately, the Orbix implementation we were working with did not permit examination of the arguments by our interceptor.

exercised in a manner similar to its intended use. LPA Vision is driven by a database of parts and other planning artifacts. In these experiments, we used realistic databases provided by LPA Software. Time-stamped records of messages sent and received were written to files and later analyzed offline.

We also simulated attacks on LPA Vision. We designed two scenarios for plausible attacks on the program, in consultation with LPA Software. The scenarios, which were intended to be as realistic as possible, describe a malicious insider taking specific “planning” actions that would ultimately cause harm to the company if left uncorrected. In the first of these, the perpetrator uses the standard LPA Vision interfaces to do this, after (we imagine) discovering an unattended terminal already logged in and running the software.

In the second scenario, the perpetrator causes a very similar form of damage, using a “rogue client.” A rogue client is one that an attacker writes to make arbitrary requests of the back-end database. The rogue client can directly invoke operations on the internal objects of the application; it thus circumvents program logic embedded in the legitimate client.

The other CORBA application that we used in our experiments is the *PersonnelTracker*. This distributed program is a communication tool that gathers information from employees about their location or status (e.g. “in the office,” “at lunch”) and makes the information available to other employees. It was written at ORA to demonstrate the CORBA Immune System, and it represents a broad class of applications in which a front end interacts with the user and updates a back-end database. Security concerns with this program, which form the basis for possible “attacks,” are

- Privacy—users control who can see their status
- Integrity—preventing malicious alteration of a user’s status

Users of the application must log in with an application password to change their own status or to view the status of others. The application relies on this authentication for access control to maintain privacy and integrity.

ORA employees used the *PersonnelTracker* application intermittently over a period of several weeks, as data was collected. During the training period, the application was not secretly “hacked” or abused. After the training period, a rogue client program was used to attack the server. We chose a sample of traces (records of connections) from the normal data to use as official training data. After eliminating some uninteresting polling requests, the training data represents 171,673 requests from 49 traces. Other normal traces were used to test for false positives. We checked for false negatives in interactive sessions (not recorded). For any sliding window of width greater than 1 (see below), it was easy to set a threshold value that could distinguish between normal and rogue traces.

In both experiments, we used the traces obtained during training to define the self database (or “normal”). The attacks were then compared with the self database to check for misses. Finally, additional normal traces—not part of the training data—were compared with the self database to check for false alarms.

### **3.3 Results**

The attacks that we mounted on the candidate applications fell into roughly two categories: rogue clients and simulated abnormal use of application clients. The rogue clients are representative of direct attacks on the application database, attacks that are made feasible by the distributed object model for applications. Abnormal use of application clients is more like traditional attacks on privileged Unix processes: the attacker attempts to subvert the unaltered application. Such attacks may involve “stolen” passwords (an unauthorized user acting in the place of an authorized one) or unusual activity (e.g., multiple login attempts, possibly in order to guess a password).

We can summarize our results as follows:

- Our candidate definition of self for CORBA applications was able to detect some kinds of rogue clients. Even when a programmer familiar with the application tried to imitate the application in a rogue client (in order to confuse the intrusion detection algorithm), the algorithm could detect differences.
- We were unable to obtain conclusive results for misuse attacks. The LPA Vision application is very large, and we were unable to perform enough training to “cover” normal usage. There is no concept of misuse for the PersonnelTracker. No misuse attacks are included here.
- The choice of window size is a matter of convenience. It does not seriously affect the ability to detect intrusions. For very small window sizes, the false negative rate is high, but once a certain minimum value has been reached, increasing window size merely increases the length of time required to achieve coverage.

#### **3.3.1 The PersonnelTracker experiment**

##### **3.3.1.1 Window size ( $N$ ) and coverage**

Aware of Stephanie Forrest’s work with Unix processes, we first tried sliding window widths between 6 and 12. We found that for the PersonnelTracker application, a width of 2 was adequate to avoid most false alarms, and 3 to eliminate them. These values are much smaller than Forrest obtained for Unix processes, no doubt because of the coarser granularity of CORBA

object requests. We suspect that values in this range would generally apply to CORBA applications.

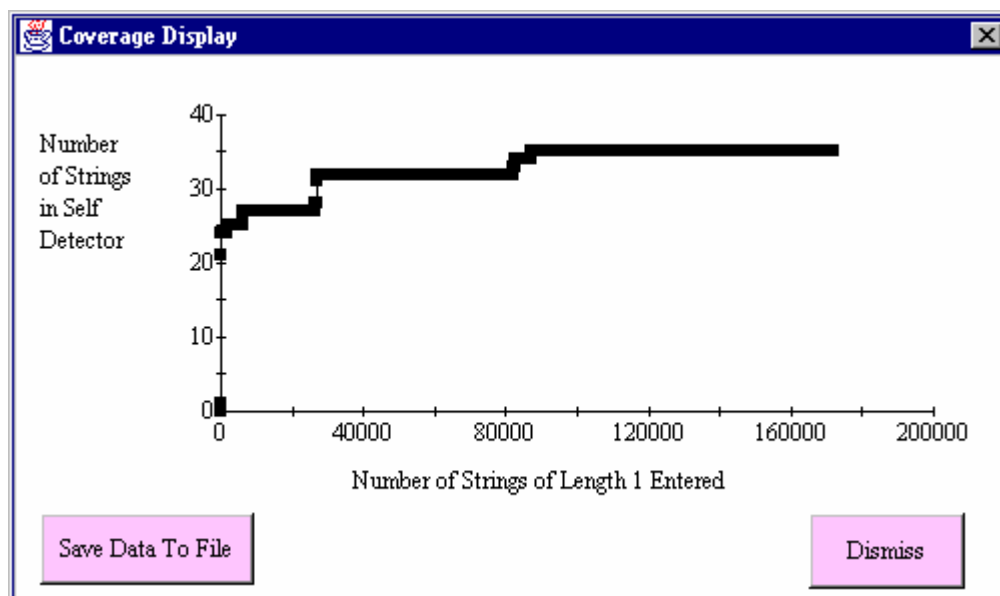
We believe that the self database naturally consists of many strings of different lengths, corresponding to fragments of the client code (fragments of straight-line code connected by conditionals, loops, and subprogram calls). Using a fixed  $N$ , which is convenient for detection, can only roughly approximate this natural database. If we pick a large value for  $N$ , relative to the natural database, then we are really trying to detect *pairs* of the strings in the natural self database. The result is rapid growth in the size of the (approximate) self database—or, alternatively, the inability to achieve coverage.

Table 1 shows the size of the self database as a function of  $N$ . Note that a window width of one simply records the different possible requests.

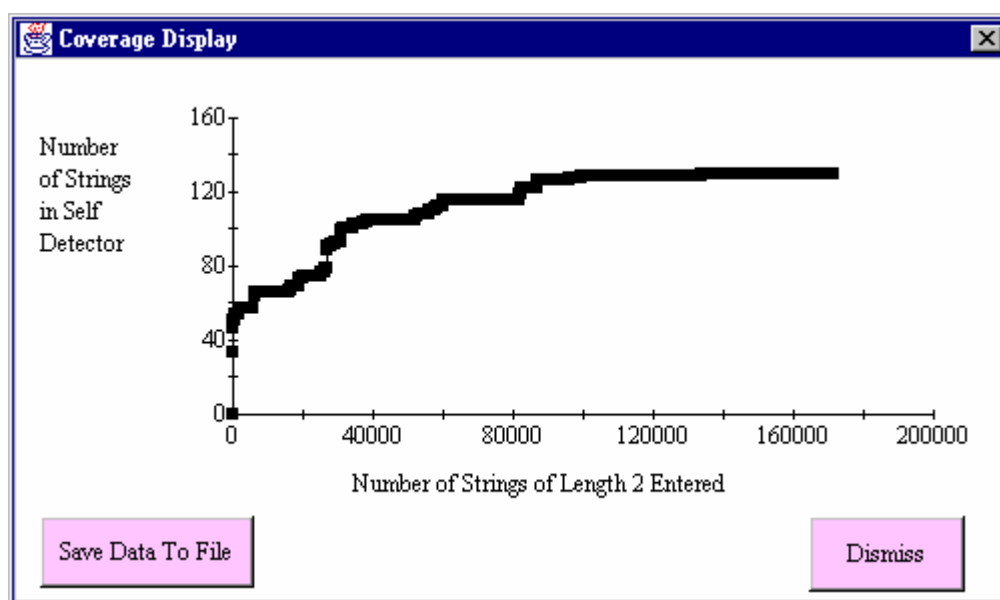
<b>N</b>	<b>Size of self database</b>	<b>Coverage?</b>
1	35	Yes
2	130	Yes
3	256	Yes
4	385	?
5	504	No
6	614	No

**Table 1. Size of self database as a function of window width.**

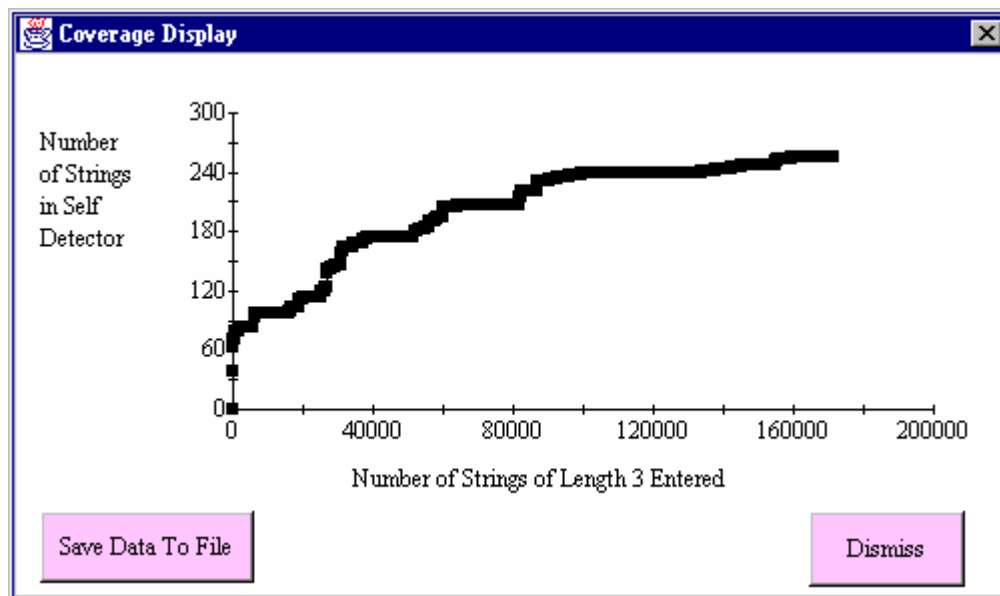
Figure 1 through Figure 4 show the effect of window size ( $N$ ) on our ability to achieve coverage. Each graph shows the growth of the number of distinct self database values as requests in the training data are processed. For example, for  $N = 2$ , the graph shows that the self database quickly reaches a size of 60, but does not reach size 120 until about 100,000 requests in the training data have been processed. Notice that for values of  $N$  that are “too small” (1) coverage is achieved quickly (at the expense of false negatives).



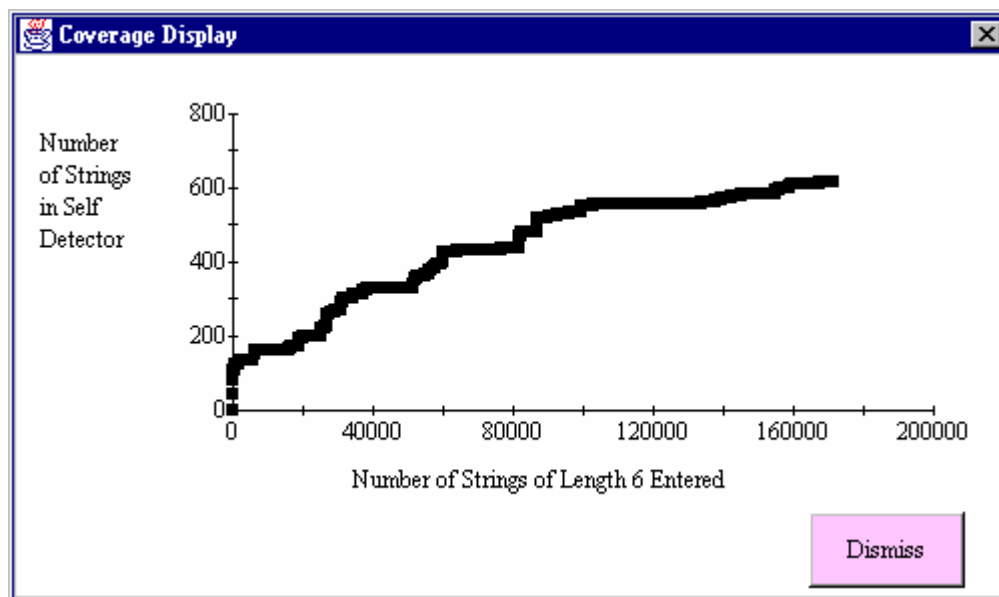
**Figure 1. Coverage for  $N = 1$ .**



**Figure 2. Coverage for  $N = 2$ .**



**Figure 3. Coverage for  $N = 3$ .**



**Figure 4. Coverage for  $N = 6$ .**

### 3.3.1.2 Anomaly values

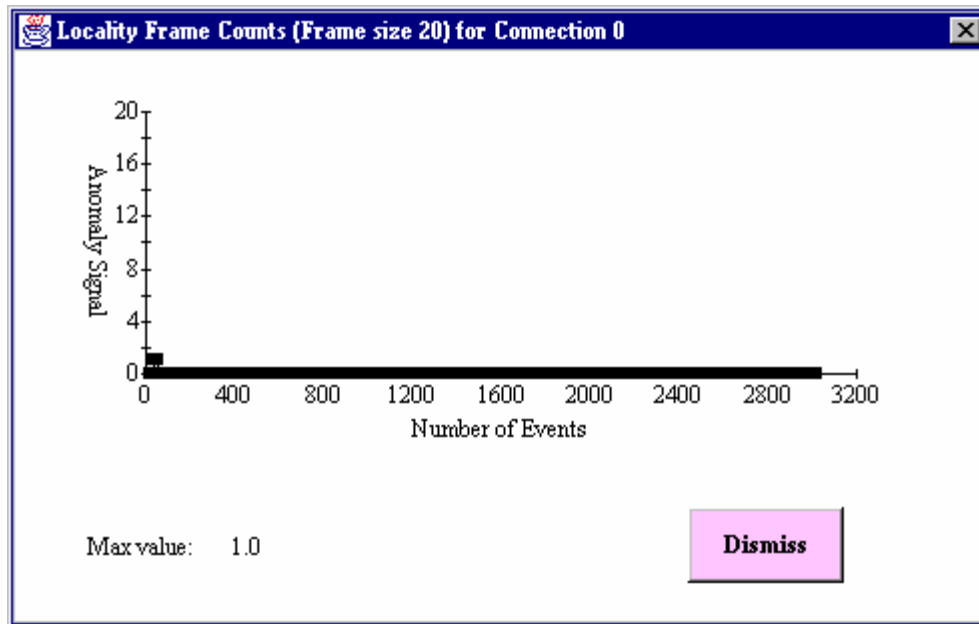
Table 2 provides an overview of results from the PersonnelTracker experiment. The traces in the table are identified by code names. For each trace, Table 2 gives the length and the minimum and maximum locality frame values. (Recall that the locality frame values gives a local measure of anomalousness, rather than an overall measure for a trace.) A locality frame of length 20 is assumed in this table. The anomaly measure used is the number of sliding window frames (out of the last 20 frames) with anomalous values.  $N$  is the width of the sliding window used.

		$N = 1$		$N = 2$		$N = 3$		$N = 6$	
Trace	Length	Min	Max	Min	Max	Min	Max	Min	Max
42.43	1445	0	0	0	0	0	0	0	0
55.37	3017	0	0	0	1	0	2	0	8
51.24	2718	0	2	0	8	0	15	0	20
ORA rogue	155	0	0	12	20	17	20	20	20
BBN rogue	27	0	0	20	20	20	20	20	20

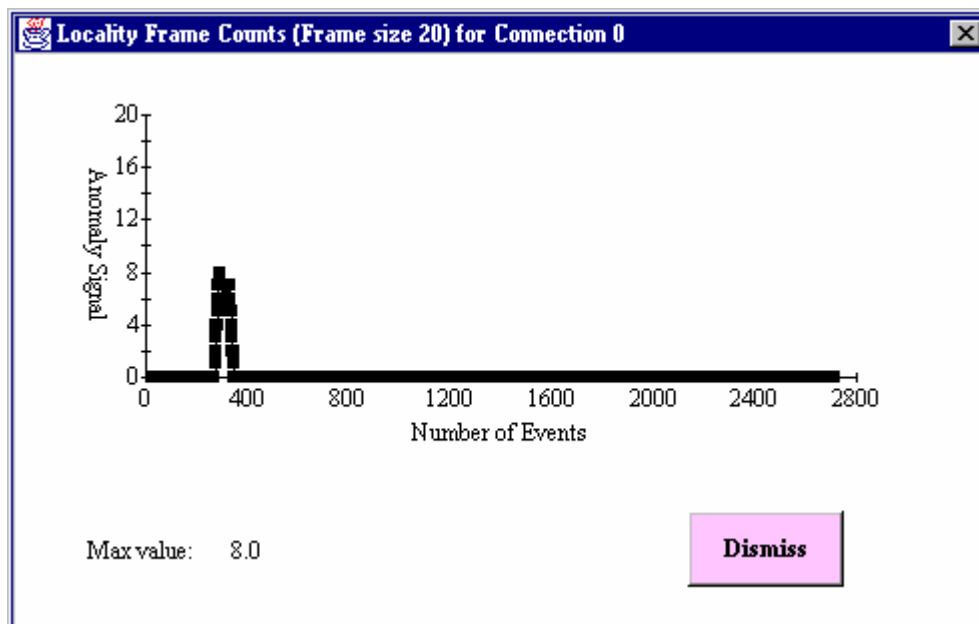
**Table 2. Summary of PersonnelTracker experimental results.**

The first three traces in the table (named after the final few digits of their timestamps) correspond to normal use of clients. The first trace (42.43), which is entirely normal, was typical of almost all traces during normal usage. In fact, it was hard to find normal traces containing abnormalities, such as trace 51.24 and trace 55.37. Anomaly graphs for traces 55.37 and 51.24 are shown in Figure 5, Figure 6, and Figure 7.





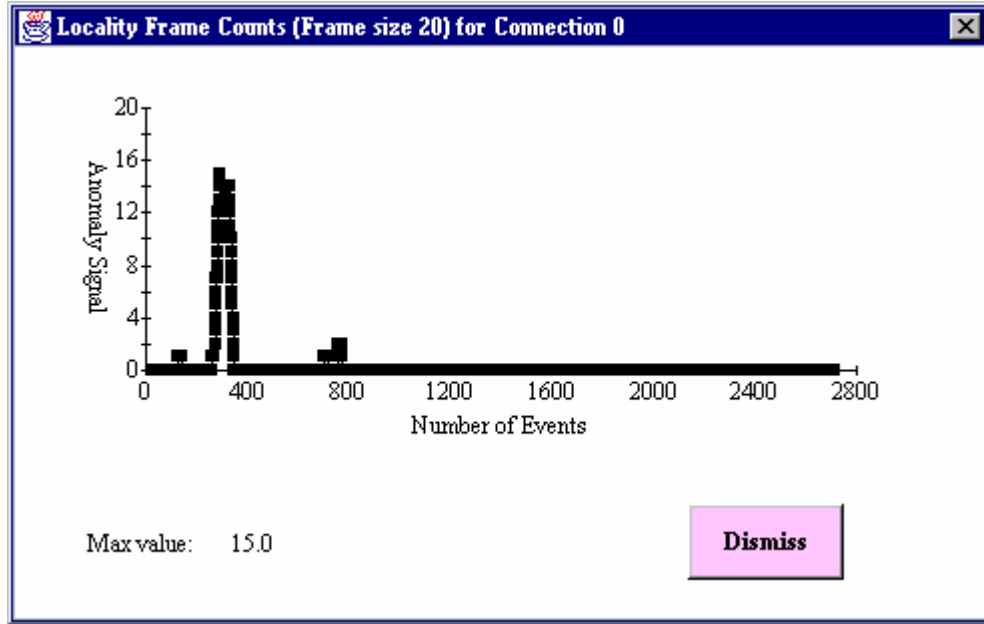
**Figure 5. Anomaly graph for 55.37,  $N = 2$ .**



**Figure 6. Anomaly graph for 51.24,  $N = 2$ .**

Trace 51.24 deserves special mention. This trace showed the greatest deviation from the training data of our “normal” traces. It was the only normal trace to achieve the maximum anomaly value. The maximum anomaly value of 2 for  $N = 1$  implies that two of its requests do not appear

anywhere in the training data. Careful examination of the trace shows that it contains two instances of the request “setPrivacy.” Evidently, none of the users set access rights for their data in the traces that were selected as training data! For any  $N$  greater than two, this trace is apt to generate a false alarm (see Figure 7).



**Figure 7. Anomaly graph for 51.24,  $N = 3$ .**

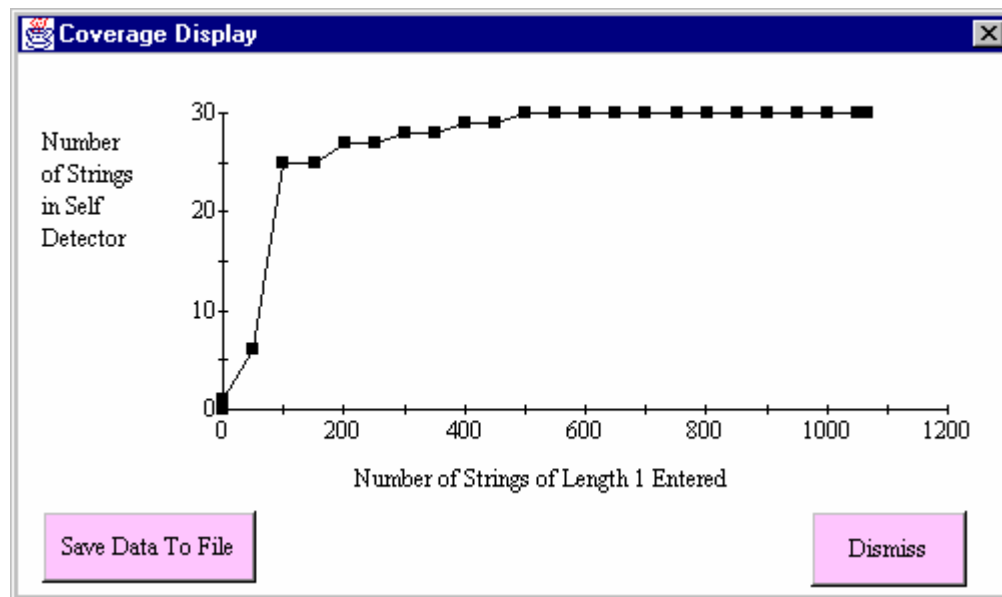
The last two traces are from rogue clients. Most of our experimentation with rogue clients used interactive clients. We wrote one rogue client program, which appears in Table 2. Summary of PersonnelTracker experimental results.. Mike Dean of BBN kindly supplied us with another rogue client, which he prepared using the Interface Description Language (IDL) specification of the PersonnelTracker; this rogue is called the BBN rogue in the table. Both rogue clients are unequivocally anomalous.

Table 2 shows the effect on anomaly values of varying  $N$ . Note that  $N = 1$  corresponds to checking that all requests made by the client appear in the training data. As expected, anomaly values rise monotonically with increasing  $N$ . What is more surprising is that a sliding window width of 2 is sufficient, for this example, to distinguish between normal and rogue traces.

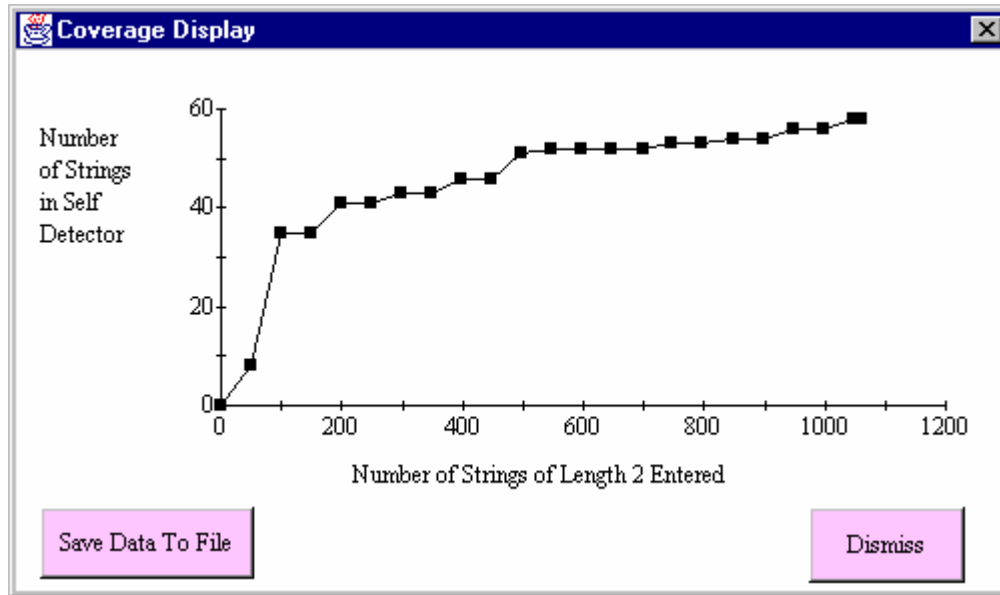
### 3.3.2 The LPA Vision experiment.

The LPA Vision experiment was much smaller than the PersonnelTracker experiment. The training data comprised only a few traces, and only two

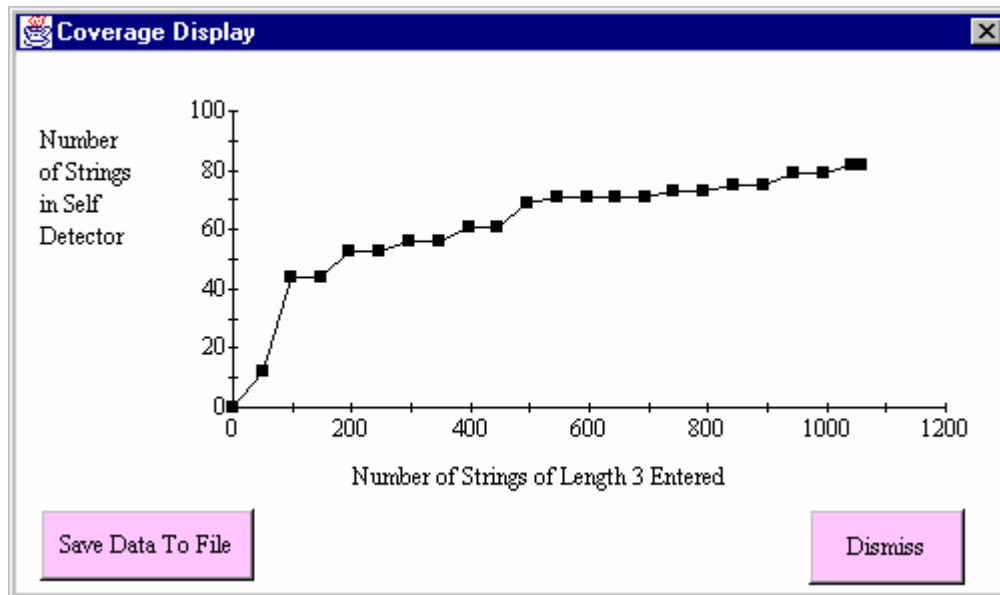
attacks were simulated. The coverage graphs are shown for values of  $N$  equal to 1, 2, and 3, in Figure 8, Figure 9, and Figure 10, respectively. The  $N = 1$  graph shows good coverage, however, that small a value of  $N$  yields a high number of false negatives. As  $N$  increases to 2 and 3, the coverage drops off, as the number of strings in the self detector continues to grow. For higher values of  $N$ , the number of strings in the detector continues to increase and the overall coverage decreases.



**Figure 8. Coverage graph for  $N = 1$ .**



**Figure 9. Coverage graph for  $N = 2$ .**



**Figure 10. Coverage graph for  $N = 3$ .**

The results from the two attacks are summarized in Table 3. The rogue client attack was 100 percent anomalous—in other words, easily detected by

our method.<sup>3</sup> The attack that simulated using a standard client with a stolen password was only 1 percent anomalous. We were not able to test the self database against additional normal traces, hence we have no information about false positives. All traces were used to generate the self database, which includes only a little over 1,000 strings. We do not believe that the training data is even close to covering normal use. In Table 3, Scenario 1 is the attack that attempts to use the standard application front end for unusual actions.

		$N = 1$		$N = 2$		$N = 4$		$N = 6$	
Attack	Trace length	Min	Max	Min	Max	Min	Max	Min	Max
Quick Approve	178	0	0	0	0	0	2	0	3
Rogue	17	0	2	5	5	5	5	5	5

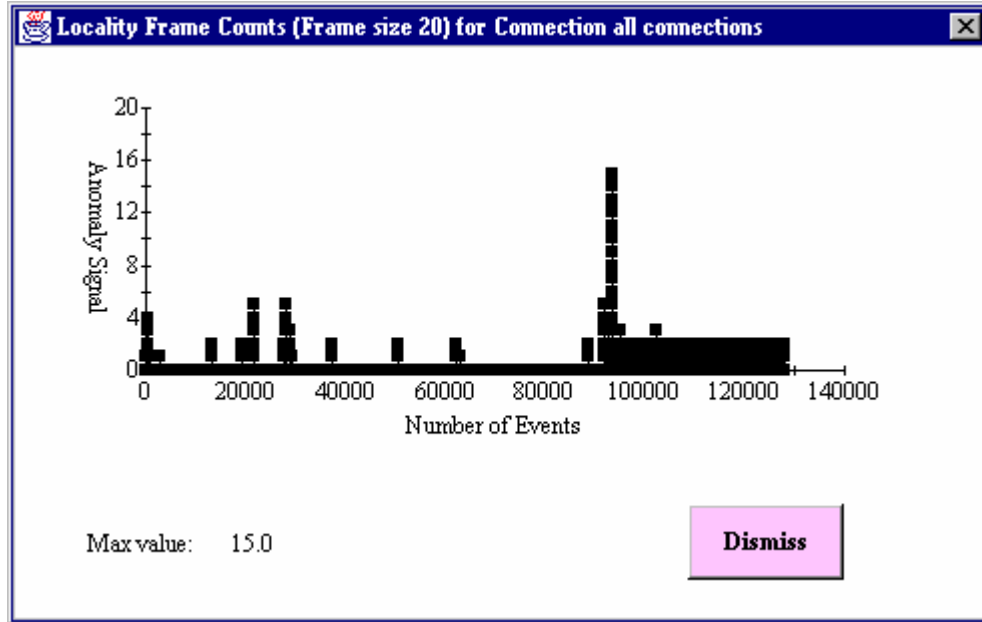
**Table 3. Results of the LPA Vision experiment.**

### 3.3.3 Detection efficiency and false-positive rate

The experimental results described so far show that the CORBA Immune System has a high detection efficiency (that is, that it does not miss attacks). To estimate its false positive rate, we tested 43 normal traces that were not used as training data. The results are shown in Figure 11. The spike in that figure corresponds to the spike of Figure 7, and is explained by the fact that the training data does not contain operations related to privacy.

---

<sup>3</sup> Unfortunately, this appealing result is invalidated by the fact that our training data did not nearly cover normal usage.



**Figure 11. Anomaly graph for normal traces.**

While much more data would be required to compute a definitive false-positive rate, the results we have obtained strongly suggest that it is possible to obtain a very low false-positive rate using this method.

### 3.4 Discussion

The results that we obtained are still preliminary. We were able to test our candidate definition of self on only two applications, one of which was developed in-house, and the other of which was too large for us to obtain coverage. While these are representative of a broad class of applications, we need to gain more experience with a variety of distributed applications.

Previous work on Unix benefits from the existence of well-known attacks from actual system use. CORBA applications are too new for any library of known attacks to be available—certainly not attacks on the applications themselves. Therefore we had to content ourselves with simulated (although realistic) attacks.

An ideal experiment would test the detection algorithm against data from a real site, either relying on chance to provide some intrusions or injecting some test intrusions into the site. The alarms raised by the detection algorithm could then be compared with “ground truth” to measure detection efficiency and false alarm rates. Unfortunately, most companies and organizations are not willing to become experimental subjects for security

experiments because they perceive very little advantage and considerable risk.

Intuitively, one might expect that window size for sequences of CORBA requests should be shorter than for, say, Unix kernel calls, because of the difference in granularity. We found that window size makes little difference in catching rogue clients, but it certainly affects the cost of obtaining coverage. In general, the window size should be the smallest possible that is able to distinguish between normal activities and attacks. To make an intrusion detection system practical, we must find a way to automatically ascertain the minimum window size that distinguishes between normal usage and attacks.

## **4 Research at the University of New Mexico**

Over the past year, the UNM group focused on three activities: Collecting new data sets, studying alternative data analysis methods for the system-call data (based on these new data sets), and developing a real-time monitoring tool for Linux. Most of this work was funded under another DARPA grant. However, it was informed by our ongoing collaboration with ORA, and we charged the ORA grant for the time we spent communicating the results and discussing the project. The next two subsections describe our progress and give an overview of the results.

### **4.1 Data Modeling**

In this project, we had the goal of answering the following questions: What properties of the system-call data make it appropriate for the sliding window method that we have used so successfully? Are there other ways of modeling the data that would lead to better or more efficient discrimination between normal and abnormal patterns? If we had answers to these questions, then we would have a much better idea of what to expect when we try these techniques in other domains, such as CORBA, NT, or at the ATM level. To answer these questions, we looked at several frequency-based techniques, including Hidden-Markov Models, Markov trees,  $n$ -grams, and non-parametric methods such as those used in SRI's Emerald project.

There are two important characteristics of the approach introduced in [5]. First, it identifies a simple observable (short sequences of system calls) that distinguishes between normal and intrusive behavior. This observable is much simpler than earlier proposals, especially those based on standard audit packages, such as SunOS's BSM. Second, the method used to analyze, or model, the sequences is also much simpler than other proposals. It records only the presence or absence of sequences; it does not compute

frequencies or distributions, or identify which sequences are most important. The advantage of such a simple approach is computational efficiency, but the question naturally arises of whether more accurate models of the data might be possible.

Over the past several years, many statistically based learning techniques have been developed. Several such methods have the potential for generating more accurate and/or more compact models of the system-call data, and at least two groups have published results of their own experiments on alternative models applied to system calls. Most of the available methods, however, were designed for specific applications, and each has its own idiosyncrasies. Our goal was to compare these various methods as systematically as possible across a larger and more realistic suite of data sets than has been used in the past. This was not an easy task.

We selected four methods for careful study: sequence time-delay embedding (stide) which is the UNM-developed sliding window technique, a frequency-based modification of stide, which we call "tstide," a rule-induction method developed at ATT and adapted by Lee and Stolfo for the system-call data (called RIPPER), and Hidden Markov Models (HMM).

A machine-learning approach to the system-call data would construct a finite state machine to recognize the "language" of the program traces. There are many techniques for building either deterministic or probabilistic automata for this sort of task. These methods generally determine the frequencies with which individual symbols (system calls in our case) occur, conditioned on some number of previous symbols. Individual states in the automaton represent the recent history of observed symbols, while transitions out of the states indicate both which symbols are likely to be produced next and what the resulting state of the automaton will be. Many, but not all, of the algorithms for building these automata are based on the assumption that the data are stationary. A particularly powerful finite state machine is the hidden Markov model, used widely in speech recognition and in DNA sequence modeling. HMMs are computationally expensive, but very powerful. There is a great deal of information available on them, and their usefulness has been demonstrated in many areas. For these reasons, we decided to use HMMs as the finite state machine representative for our experiments.

The original studies of the system-call approach were conducted primarily on synthetic data sets. Synthetic traces are collected in production environments by running a prepared script; the program options are chosen solely to exercise the program, and not to meet any real user's requests. Although the earlier studies on synthetic data sets were suggestive, they are



not necessarily good predictors of how the methods will perform in fielded systems. Consequently, we used a wider variety of data sets for our current study. These include “live” normal data (traces of programs collected during normal usage of a production computer system), different kinds of programs (e.g., programs that run as daemons and those that do not), programs that vary widely in their size and complexity, and different kinds of intrusions (buffer overflows, symbolic link attacks, Trojan programs, and denial-of-service). We used programs that run with privilege (with one exception), because misuse of these programs has the greatest potential for harm to the system. All of these data sets are publicly available and carefully described at <http://www.cs.unm.edu/~immsec/data-sets.htm>. Intrusions were taken from public advisories posted on the Internet. We tested each of the four data-modeling methods on each of the data sets (traces of Unix programs) at several different sensitivity thresholds. False positives were computed for normal data not used during training, and true positives were computed for traces of anomalous behavior.

We compared four methods for characterizing normal behavior and detecting intrusions based on system calls in privileged processes. Each method was tested on the same suite of data sets, consisting of different types of programs and different intrusion techniques. On this test suite, three of the four methods performed adequately. Hidden Markov models, generally recognized as one of the most powerful data-modeling methods in existence, gave the best accuracy on average, although at high computational costs. Surprisingly, the much simpler sequence time-delay embedding method compared favorably with HMMs. We conclude that for this problem, the system-call data are regular enough for even simple modeling methods to work well. The average results indicate that it might be possible to achieve increased accuracy with HMMs, provided significant computational resources are available to train and run them.

However, no one method consistently gave the best results on all programs, and results between programs varied more than results between methods. Variations in false positives were due more to the complexity of the traced programs and their environments than to differences in the analysis methods. Although there are multitudes of alternative methods that were not tested, our results demonstrate that for this problem, several methods perform well. We believe that the choice of data stream (short sequences of system calls) is a more important decision than the particular method of analysis.

Historically, many computationally sophisticated methods have been applied to the intrusion-detection problem, yet there are few well-accepted solutions in widespread use. One lesson from this study is that perhaps a disproportionate amount of attention has been directed to the data-modeling

problem, and that equal attention should be paid to considering what are the most effective data streams to monitor.

## **4.2 Real-Time Monitoring and Response**

We developed a real-time monitoring system for collecting system-call data under Debian Linux 2.1. The current system performs basic online monitoring using the “look-ahead pair” method of analysis. This method was described in [5] and differs slightly from the complete sequence analysis that we have used for our offline studies. Although complete sequence analysis seems to give more precise discrimination, the look-ahead pairs method is much more efficient to implement in the kernel. Although our new kernel cannot yet load and save profiles reliably (needed for the next step of our research), we have used it over the past six months to collect several new data sets for offline analysis.

Preliminary measurements on the monitor show no perceptible impact on the performance of the computer on which it runs. This is a big result because it suggests that it is practical to do both the system-call monitoring and the analysis in real time. We have had to take a few short cuts to get everything running, so the system is not yet ready for widespread distribution.

We are developing an online monitor as the first step in implementing a version of our IDS that runs in real time. Once that hurdle is crossed, our next effort will be directed toward adapting the code for automated response. We spent much of the past six months considering the automated response problem, and we feel that thinking about automated response from the system-call perspective is a good starting point. Automated response is a controversial topic because of two factors: (1) False positives seem to be an inescapable fact of life, and (2) It seems imprudent to allow a computer to automatically take drastic measures when there is even a small probability that it is a false alarm. We believe that the flaw in this analysis is with the term “drastic measures.” We believe that a system capable of making automated responses should be taking actions that are of a small grain size, so that a few false actions are not lethal to the system. For this reason, we believe focusing on the execution of system calls is a good place to begin studying the automated response problem.

In conclusion, we believe that over the past several years, the field has made enormous progress in its ability to automatically detect the presence of intrusive and abnormal behavior. We believe that it is unrealistic to expect perfect discrimination (100% true positives and 0% false positives), and that the numbers currently being reported compare favorably with discrimination problems in other fields. Thus, we believe that the most important next step

in IDS research is to develop automated response methods that can perform well, given current detection abilities.

### **4.3 Papers Published**

[3] D. Dasgupta and S. Forrest, “Artificial Immune Systems in Industrial Applications,” accepted for presentation at the *International conference on Intelligent Processing and Manufacturing Material (IPMM)*, Honolulu, HI (July 10-14, 1999).

[7] S. Hofmeyr and S. Forrest “Immunity by Design: An Artificial Immune System.” *1999 Genetic and Evolutionary Computation Conference (GECCO)* (in press).

[16] C. Warrender, S. Forrest, and B. Pearlmutter, “Detecting intrusions using system calls: Alternative data models,” *1999 IEEE Symposium on Security and Privacy* (1999).

[8] S. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion detection using sequences of system calls,” *Journal of Computer Security*, Vol. 6, pp. 151-180 (1998).

## **5 Software Design and Development**

We designed and built a prototype CORBA Immune System to analyze interobject message traffic in a CORBA application and flag anomalous activity in near real time. The prototype implementation uses the sliding window definition of self described in Section 3.1, but the design is applicable to a wide range of definitions of self, including different methods of selecting events, different projections, and different comparison algorithms.

We also constructed a tool, called Immune Data Analyzer (IDA), to aid in analyzing experimental data and to generate descriptions of normal application behavior.

The design of the two tools is described in [11]. (Although some details have changed, the overall design has not.) Here we describe some highlights of the design. Section 5.1 describes the CORBA Immune System architecture. Section 5.2 describes the IDA architecture.

### **5.1 The Design of the CORBA Immune System**

In this section, we describe the operation and design of the CORBA Immune System. Details can be found in [11]. Further information on how to use the system can be found in [1].

### 5.1.1 What the system does

The CORBA Immune System has four modes:

- **Inoperative** In inoperative mode, the CORBA Immune System does nothing. The application can run as if the CORBA Immune System were not installed.
- **Training** In training mode, the CORBA Immune System collects data about the message traffic between clients and servers. The data can later be used to generate a description of normal behavior.
- **Test** In test mode, the CORBA Immune System displays information about every client/server connection. (Note that in the original architecture report, test mode was called experimentation mode.)
- **Detection** In detection mode, the CORBA Immune System displays warnings about anomalous client/server connections.

The Installation Guide [1] describes how to use the CORBA Immune System. Briefly,

- The application developer decides which servers to instrument. He links the CORBA Immune System's Reconnaissance/Analysis module with those servers.
- The application installer runs the CORBA Immune System in training mode to collect data about normal usage of the application.
- The application installer runs the IDA tool to generate a self detector.
- The application installer runs the CORBA Immune System in detection mode, telling it where to find the self detector. The CORBA Immune System displays information about anomalous connections, which may indicate that an attack is occurring.

### 5.1.2 Overview of how the system works

The Common Object Request Broker Architecture (CORBA) [2, 12], promulgated by the Object Management Group (OMG), provides standards for linking applications and objects across machine boundaries in a heterogeneous, networked environment. In the CORBA environment, we can speak of three layers: the underlying operating system(s), the Object Request Broker, and the application. A CORBA application is written in terms of communicating objects. The ORB architecture defines how requests are sent from *clients* to *servers* and how replies are returned.

As described in Section 3.1, the CORBA Immune System analyzes message traffic between CORBA application clients and servers, using the sliding window definition of self. We call the combination of client and server a *connection*; different clients interact with a server via different connections.

The CORBA Immune System collects a sequence of requests, called a *trace*, for each connection from a client to a server. It then compares the trace to

the self-value (viz., the set of sequences of length  $M$ ) for that type of server, using the sliding window algorithm. (The self-value has been computed from training data.) We expect that even when no attack is being made on the application, some deviation from normal may occur, typically because the training data does not cover all normal activity. To avoid false alarms, we report a possible attack only if the deviation from normal exceeds a certain (configurable) threshold.

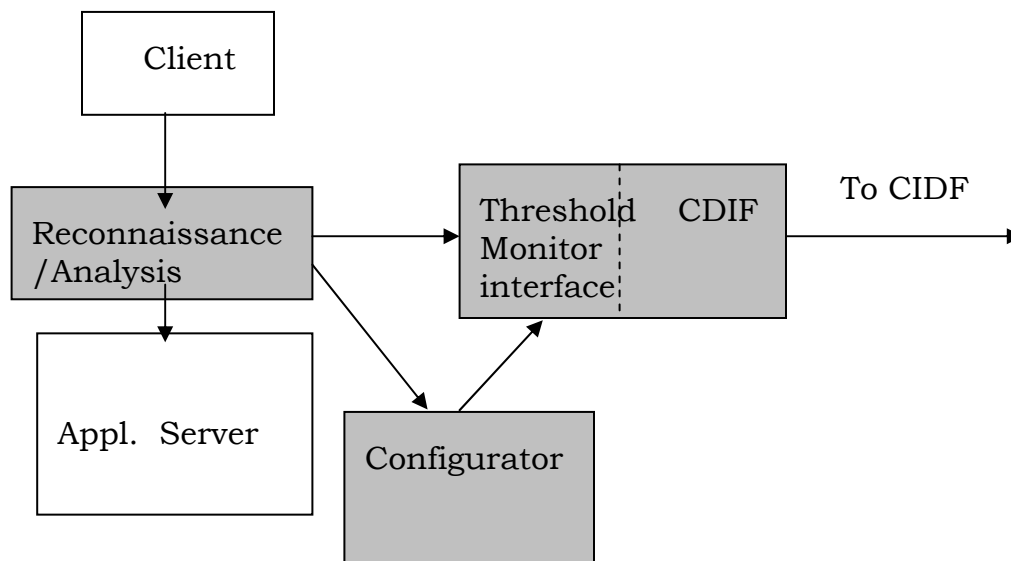
The prototype implementation of the CORBA Immune System works with Iona's Orbix, a widely used ORB implementation. However, the concepts underlying the CORBA Immune System are independent of the operating system and the ORB.

### 5.1.3 Implementation overview

The CORBA Immune System is implemented as three modules:

- Reconnaissance/Analysis module
- Threshold Monitor
- Configurator

Figure 12 illustrates the parts of the CORBA Immune System (CORBA Immune System components are shown in gray; application components are white.)



**Figure 12. Overview of the CORBA Immune System.**

The *Reconnaissance/Analysis Module* is interposed between the client and the object server. Every incoming request on a connection causes the

Reconnaissance/Analysis Module to be invoked. In training mode, the Reconnaissance/Analysis Module collects data about normal message traffic on connections. We call this information *training data*.

In detection mode, the Reconnaissance/Analysis Module plays a central role. It analyzes each request in the context of previous requests; the result is an anomaly measure for the current sequence of requests on this connection. Periodically, the Reconnaissance/Analysis Module reports to the Threshold Monitor an overall anomaly measure for each connection.

In the prototype implementation of the CORBA Immune System on Orbix, the Reconnaissance/Analysis Module is implemented as a *filter* on the server.<sup>4</sup> Orbix enables the application authors to define filters that can observe requests on the message path from client to server. The filter is awakened each time a request arrives at the server. The application developer controls which servers the system should monitor. To enable the filter, the application developer declares and initializes a C++ filter object (using the C++ filter class that is part of the CORBA Immune System) in the server source code. Adding a filter to a server requires adding two lines of code to the server, recompiling the server, and relinking.

The Reconnaissance/Analysis Module includes two important submodules:

- The *self detector* is a finite state machine that implements the notion of normal behavior for the server connection. Specifically, the IDA tool (see Section 5.2) constructs a table describing the finite state machine based on the training data. The CORBA Immune System reads that description during execution.
- The anomaly meter, which aggregates the output of the self detector to create an overall measure of how anomalous the connection is.

The second major component of the CORBA Immune System is the *Threshold Monitor*, which has two tasks:

- It receives and examines the anomaly measures of connections and alerts the operator if a certain threshold is exceeded.
- It optionally reports anomalies to a larger intrusion detection framework such as CIDF (see [14] for a report and the draft standard) or a network management system. Currently, it generates an alert using Boeing's IDIP format, which Boeing's software then translates into a CIDF message.

---

<sup>4</sup> Filters are a feature of Orbix not currently supported by the CORBA standard. However, a similar feature, called interceptors, appears in the CORBA Security Specification Version 1.2 [13]. The OMG may add interceptors to the general CORBA standard in the future.

The third major component of the CORBA Immune System is the *Configurator*, which is concerned with operational issues, such as how the intrusion detection system is started, controlled, and shut down. The operator communicates with the CORBA Immune System via the Configurator.

The Threshold Monitor and the Configurator are implemented as CORBA objects, and communication between them and the Reconnaissance/Analysis Module occurs by means of CORBA messages. The CORBA Immune System does not include special mechanisms for security or encryption; we expect it to exploit the security features of the underlying ORB.

#### **5.1.4 Finite-state machine implementation of the self database**

In [5], Forrest defines the self of Unix processes in terms of a sliding window (of constant width  $M$ ) over the sequence of system calls. Self is the set of strings of length  $N$  that appear in “normal” traces. In this section, we show how to derive a deterministic finite state machine (FSM) implementation of the sliding window algorithm. Previously, such an implementation was thought to be impractical, because it could lead to a state explosion (see [6]).

One potential cause of a state explosion is that a practical implementation must consider what state to go to in the case of an unexpected input (that is, an anomalous input). We solved that problem by introducing special “none-of-the-above” transitions in the FSM construction presented here. As can be seen from the construction, the number of states is equivalent to the number of items in an efficient representation of the sliding window self database.

We first briefly review the sliding window algorithm. Recall that in the sliding window algorithm, we slide a window of constant width across a string of input data over a constant alphabet. Consider a string of training data:

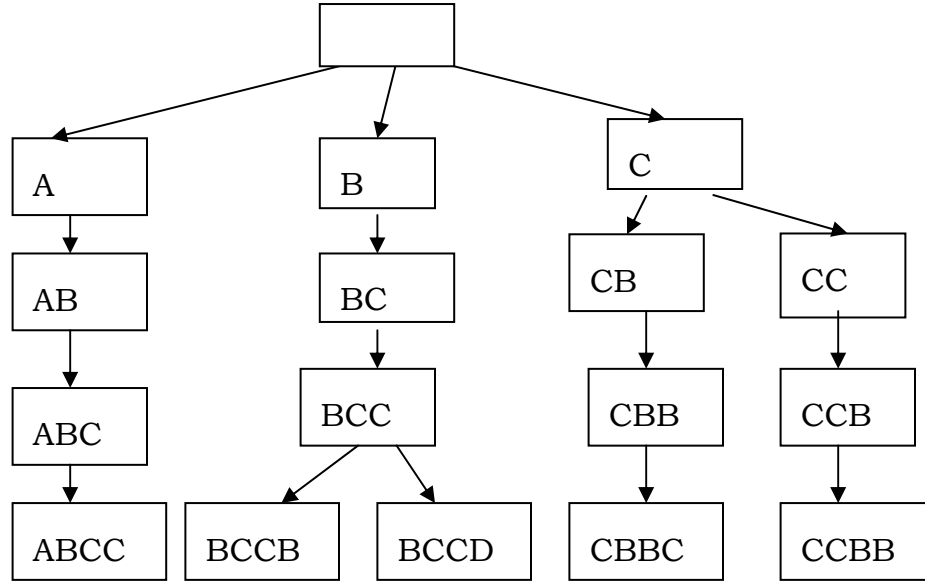
A B C C B B C C B B C C D

Then the self database consists of the set of all substrings of, say, length 4 of the training string:

ABCC  
BCCB  
CCBB  
CBBC  
BBCC  
BCCD

We can represent the self database as a tree of depth  $N$ . Each node at depth  $k$  represents the first  $k$  letters of a substring. Figure 13 shows part of the tree for the example training string (the branch representing BBCC is omitted, to simplify the figure). Thus, the nodes of the tree represent all legal substrings of the training string.

Each transition may be labeled with the letter that leads to the child substring.



**Figure 13. A partial tree for the example string.**

Given such a tree representation of the self database, a straightforward implementation of detection using the sliding window algorithm requires descending the tree from the root to depth  $N$  for each new window value. Suppose, for example, that the current window contents are BCCB. The detector would descend the tree, passing from the root node through B, BC, and BCC, to BCCB.

An alternative implementation maintains  $N$  pointers into the tree. In this implementation, if the current window content is BCCB, there are pointers to BCCB, CCB, CB, and B. When a new input symbol is read, each pointer is updated. For example, if the next letter is B, the pointer to BCCB is replaced with a pointer to B, and the next three are updated to CCBB, CBB, and BB. Both of these implementations require  $N$  operations for each input symbol.

It occurred to us that handling a single symbol of input could be done using fewer operations, which led to the FSM implementation. Our construction implements what we call the “two-finger” algorithm, which can best be explained as follows.



Imagine that while reading the string, you enclose a substring of letters with your two index fingers. As you read the string, you move your fingers so they always contain the current contents of the sliding window. Suppose you are reading the test string “ABCCBCCB.” You begin with both fingers together at the beginning of the string. This (initial) state corresponds to the root node of the tree, or the empty string. Moving your right finger one letter to the right corresponds to descending one level in the tree of Figure 13. In this case, with successive moves of your right finger, you successively visit states A, AB, ABC, and ABCC. However, we require that your fingers be separated by at most  $N$  letters. In order to move past the  $N$ th letter, you must move your *left* finger one letter to the right. This results in the state representing the suffix of the current state’s substring. For example, from state ABCC you move your left finger and arrive at state BCC. (Note that this state is guaranteed to be in the tree.) You may now move your right finger again, which brings you to state BCCB.

The left finger moves correspond to a set of *suffix* transitions that we can impose on the tree. That is, we define a transition from each state to the state corresponding to its suffix. For example, the suffix of A is the empty string state; the suffix of state AB is state A. It is easy to see how, using left and right finger moves, we can traverse any string that contains no anomalies.

However, the sixth letter in the test string introduces an anomaly. After substrings BCCB, there are two anomalous substrings—CCBC and CBCC—that do not appear in the training data. From state BCCB, a left finger move brings us to state CCB, but there is no transition for a next letter C from that state (because CCBC isn’t in the self database). If, however, you move your left finger a second time,<sup>5</sup> you get to state CB. There is also no transition for C from state CB. Moving the left finger a third time, we arrive at state B, which has a transition to BC. The rule, then, is to move the left finger only until you can move the right finger again (i.e., until you reach a state in which a transition for C occurs).

In this example, we have been forced to take *three* suffix transitions, which corresponds to *two* anomalous substrings in the test data (CCBC and CBCC). In general, the number of anomalies for each right finger move is the number of left finger moves minus one (because one left finger move is always required). Now we can traverse any test string, including anomalous

---

<sup>5</sup> We can also call the suffix transition the “none-of-the-above” transition. It is the default transition, which we take when none of the right-finger transitions is possible. Note that suffix transitions do not consume their input. However, the FSM that we have defined is deterministic.

ones, and we can count the number of anomalies that occur during the traversals.

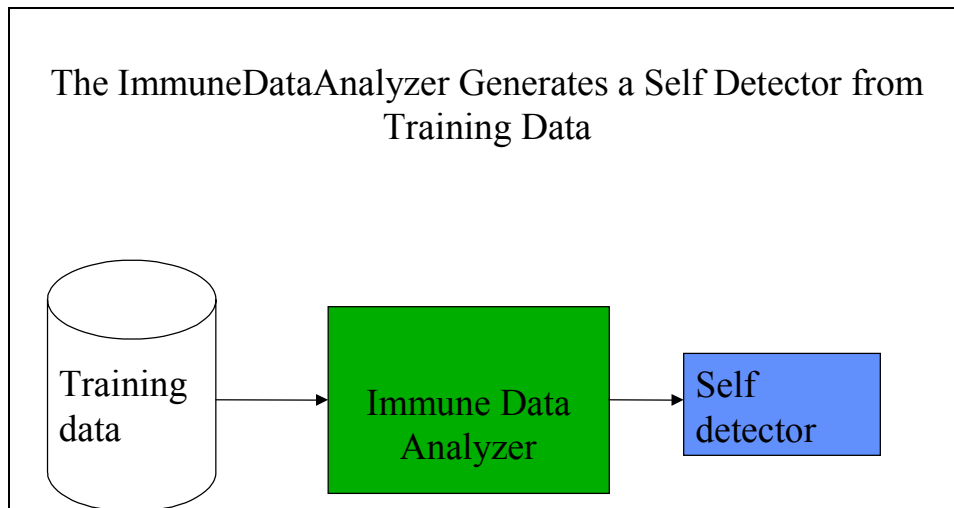
Suppose that a new letter appears in a test string, one that does not occur in the training data. To accommodate that possibility, we add one more state, the UNKNOWN\_SYMBOL state, as a child of the root. This completes the construction of the FSM for the two-finger version of the sliding window algorithm.

We have implemented the two-finger algorithm in the Immune Data Analyzer. By adding the suffix transition for each node at the time the node is created, we construct the self database directly as a finite state machine.

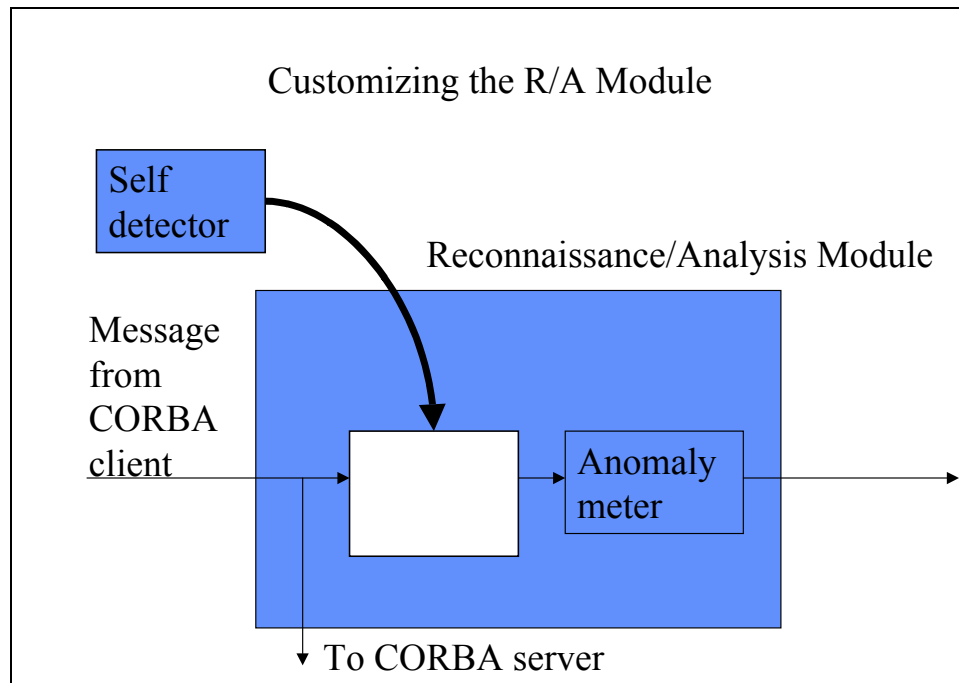
The self database may be very large, including tens, hundreds, or even thousands of strings of length  $N$ . Nevertheless, the finite state machine executes just two transitions for each input symbol, plus one transition for each anomalous input string, so the computational burden associated with each input symbol is low.

## 5.2 The Design of the IDA

The IDA is a tool for building self detectors from training data describing the normal operation of a CORBA application, as shown in Figure 14. After the self detector is generated, it is used as a custom component as part of the reconnaissance/analysis module, as shown in Figure 15. The self detector evaluates incoming messages from the CORBA client and reports anomalies to the anomaly meter.



**Figure 14. Self-detector generated by the IDA.**



**Figure 15. Self-detector and the R/A Module.**



**Figure 16. User interface of the IDA.**

Figure 16 illustrates the IDA, which is constructed as a sequence of four tab panels:

- **Input Training Data.** In the first tab panel, the user inputs the training data from a file.
- **Generate Self Detector.** In the second tab panel, the user can generate a self detector. Currently only the sliding window definition of self is supported, but the user can choose the length of the window. The tool provides a graph estimating how well the training data covers normal usage (see Section 3.1 for a brief discussion of coverage).
- **Save Self Detector.** In the third tab panel, the user can generate a description of the self detector as a finite state machine.
- **Test Self Detector.** The user can test the generated self detector against new traces. The IDA produces anomaly measures for each new trace. Several different algorithms are available to determine the anomaly measures.

The user interface is presented in detail in the Installation Guide [1].

## **6 CIDF Promotion to the Government and Commercial Sectors**

Over the last two years, ORA has contributed to the standardization of a common framework for intrusion detection known as CIDF. ORA has contributed in the areas of architecture definition and the establishment of a connection between CIDF and the SNMP network management standards defined by the Internet Engineering Task Force (IETF). An additional contribution has been in the formulation of data content requirements for intrusion detection alert messages. In this report, however, we focus on the work done to promote CIDF to government agencies and commercial vendors.

The CIDF standard was initially defined principally by the DARPA-sponsored intrusion detection community. A year into the standardization effort, the CIDF community decided to become one of the many Internet Engineering Task Force (IETF) standards efforts. Without becoming part of a larger standards group, there was concern that the effort to standardize intrusion detection would not become a commercial reality. The goal of the ORA CIDF promotion effort was to visit the commercial vendors and bring them into the standardization effort. The specific goals of these trips were:

- educate the vendors about intrusion detection standards, using CIDF as the draft standard,
- solicit their opinion on requirements for intrusion detection standards, and
- get them involved in and committed to the standards effort.

These goals were met by visiting the vendors, promoting the standardization effort, and signing them up for IETF participation. The cooperation of the commercial groups was needed to ensure the success of this standard in IETF. The Intrusion Detection Exchange Format working group (IDwg) was launched in December 1998, and is an active part of the IETF with a charter and clearly defined goals. The companies visited as part of this promotion project were Axent, Centrax, Network Associates, CISCO, and ISS are now all active members of the IDwg.

### **6.1 Summary of Vendor Visits and Accomplishments**

Table 4 presents a list of the companies visited and the location and name of the intrusion detection commercial product of each one.

<b>Company name</b>	<b>Location</b>	<b>Product(s)</b>
Axent Technologies	Rockville, MD	NetRecon, Intruder Alert
Centrax	San Diego, CA	ENTrax
HP	Cupertino, CA	HP Openview Node Sentry
Network Associates	Santa Clara, CA	CyberCop Network, CyberCop Server, CyberCop Scanner
Newbridge Networks, Inc.	Herndon, VA	High Speed ATM
SAIC	San Diego, CA	CMDS

**Table 4. Company and commercial product matrix.**

#### **6.1.1 HP (October 5, 1998)**

Maureen Stillman of ORA presented an overview of CIDF and discussed requirements for intrusion detection standards with the Openview Security Management Products group at HP in Cupertino, CA.

This group is working on the development of a product called HP Openview Node Sentry, which will be commercially available in 1999. The product will provide a graphical user interface to a number of intrusion detection and network management systems and offers an integrated solution for enterprise security management. HP views intrusion detection standards as critical technology for this product because they want as many commercial vendors as possible to “plug in” to their management console. HP has announced a strategic alliance with CISCO and is working to form other alliances. Thus, HP has good business reasons for adopting intrusion detection standards. Our technical discussions reflected this strategic plan. They agreed to take an active role in the IETF and present their requirements for an ID standard at the IETF kickoff meeting.

#### **6.1.2 Centrax (August 5, 1998)**

Centrax was the first to sign up and commit to working on intrusion detection standards. They presented their requirements for ID standards at the IETF kickoff meeting. They have been active participants in the standardization process.

### **6.1.3 Network Associates (October 5, 1998)**

We presented an overview of CIDF and discussed requirements for intrusion detection standards at Network Associates in Santa Clara, CA. The product developed by this group, Net Tools, is a toolbox combination that includes PGP, Gauntlet Firewall, and McAfee Total Virus Defense. They requested a discussion with Stuart Staniford-Chen (IDwg co-chair) to voice their concerns and ascertain the technical direction of the group. Ultimately, they attended the IETF kickoff meeting and presented their requirements for intrusion detection.

### **6.1.4 Newbridge (November 17, 1998)**

On November 17, we visited Newbridge and made the same presentation. Newbridge responded that they would attend the kickoff meeting if they didn't have a conflict with other IETF meetings. They are heavily involved in the IETF and work extensively on ATM standards. Their business supports and relies on standards efforts.

### **6.1.5 Axent Technologies (November 16, 1998)**

On November 16, we visited Axent Technologies and Newbridge to educate them in intrusion detection standards and ask them to join us in the standardization effort. Axent was originally skeptical, but ultimately agreed to join the standardization effort and present at the IETF kickoff meeting.

### **6.1.6 CISCO (August, 1998)**

We signed up Kevin Ziese of CISCO's Austin-based security group to present their requirements at the IETF kickoff meeting. This group works on the former Wheelgroup (purchased by CISCO) commercial line of IDS products). With CISCO on board, we were more easily able to persuade many other companies to join the standard's effort. CISCO also attended the CIDF meeting in Seattle on October 20-21.

### **6.1.7 ISS**

Stuart Staniford-Chen signed up ISS to present their requirements at the IDdwg in Orlando. ISS is committed to intrusion detection standards. Mark Wood of ISS is the editor of the intrusion detection requirements document.

### **6.1.8 SAIC**

After several months of discussions with DISA, the government contract group at SAIC committed to attend CIDF meetings as well as attend the IETF kickoff meeting. Robert Thuleen is the point of contact for SAIC and Lynn Henderson claimed that DISA would fund their participation. Although SAIC

claimed that they would participate, they never came to any meetings. The group working on the CMDS toolkit was later sold to another entity.

Refer to Table 5 for a summary of vendor requirements for IDS standards.

Company	Signatures of attack	Configuration management	Console accepts alerts from any IDS
Axent Technologies		✓	✓
Centrax	✓		
HP		✓	✓
Network Associates	✓		✓
SAIC	✓		

**Table 5. Requirements for intrusion detection standards.**

With respect to the openness of the IDS and data format, and security supported by the IDS software, in general, we found that the vendors are building closed proprietary systems with minimal built-in security mechanisms. Table 6 presents a summary of the results.

## 6.2 Vendor Questionnaire

Each vendor was presented with a questionnaire concerning the openness of their intrusion detection technology and their attitude on standards. The vendors were told that the answers could not be proprietary, as this would be published as a DARPA report. The appendix contains the answers to these questions from each vendor. These questions were compiled in collaboration with Todd Heberlein of NetSquared.



<b>Company</b>	<b>Proprietary data format</b>	<b>Patents, licensed technology</b>	<b>Security services</b>	<b>Support for SNMP</b>
Axent Technologies	Data transfer in encrypted ASCII format	Yes	We use our own	Yes
Centrax	Yes	Yes	DES	Forward alerts through SNMP
HP	N/A	N/A	N/A	Yes
Network Associates	Yes	Yes	PGP, MS-CAPI and proprietary services	For alerts, generate SNMP traps
SAIC	Yes	Yes	DES	Alert message to SNMP trap

**Table 6. Openness of product and security services.**

### **6.3 Lessons Learned**

What did it really take to sign up these vendors? Here are some lessons learned for the benefit of those interested in going down a similar path.

**The Commercialization Subgroup.** We had great support during this. A CIDF commercialization subgroup was formed, comprised of Paul Proctor from Centrax, Brian Witten of (at that time) AFRL-Hanscom Site, and Dave Donahoo from CSC at the Air Force Information Warfare Center. The group formulated ideas on how to motivate commercial vendors and was especially strong in the area of making the business case to enable vendors to commit. CIDF would not have been successful without this group.

**Business justification for standards.** The commercial vendors all stressed the need for a business justification to persuade their higher management to commit. The CIDF commercialization group helped to address this issue. Ultimately, however, each vendor had to make the case for themselves with their higher management.

**Choice of representative.** It was important to send a representative that the vendors did not consider a competitor or a threat. This was critical to getting them to say where they stood on the idea of a standard.

**Getting the right people to the meeting.** To get the necessary commitments from the vendors, the right people need to be at the meetings – the best technical people, as well as marketing representatives and important decision-makers.

**Stating the business case.** The ORA representative prepared for each visit by reviewing each vendor's Web site and reading relevant papers to understand the company, their product, and their competition. The vendors were informed that ORA was visiting all of their competitors, which really shook them up, causing them to take the standard's initiative seriously.

**Get the vendor's technical interest.** The technical talk on CIDF was about an hour long and stimulated interesting technical exchanges. The vendors had clearly not given much thought to the idea of a standard (with the exception of HP), so presenting CIDF as a work in progress sparked a lively debate. This provided intellectual motivation for the technical staff.

A presentation on CIDF entitled "Common Intrusion Detection Framework Overview" was prepared and a bound book containing the following information was handed out to the participants:

- Overview of CIDF presentation
- Copy of the draft IETF charter
- Copy of a published paper on CIDF
- Copy of the questionnaire

## **7 Conclusion**

For the past two years, we have conducted research on applying computational immunology to distributed object systems. We created a definition of "self" for such systems, conducted experiments to validate that definition, and built a prototype intrusion detection system for applications build on representative CORBA middleware.

**Trusted applications and the rogue client attack.** We have defined and advertised the problem of the rogue client in applications and shown that computational immunology can detect such attacks with high efficiency and an extremely low false alarm rate. The problem of rogue clients needs to be addressed not only in an intrusion detection system, but also by access control mechanisms at the middleware and OS levels.

We have articulated the four components of a definition of self in any area and created a general tool for analyzing self data and building a self database. Our colleagues at the University of New Mexico have demonstrated of these four, the most important are focusing attention on a suitable computational entity and identifying a representative data stream that characterizes that entity. Unless a characteristic data stream has been identified, sophisticated detection mechanisms are pointless; once it has been identified, rather simple and inexpensive mechanisms will suffice. Much of the work of other groups has focused on mechanism. By contrast, ORA applied the UNM mechanism to a new problem, the rogue client attack. We designed and built a prototype CORBA Immune System to test our ideas. The CORBA Immune System successfully demonstrated its ability to detect rogue clients with high efficiency and a very low false positive rate.

We developed a particularly efficient implementation of Forrest's sliding window algorithm. The implementation uses finite state machines for computational efficiency. We are able to incrementally construct the self database as a finite state machine during training; the completed FSM can then be used for detection.

**The Unix (Linux) name daemon exploit.** As part of the work of this contract, our colleagues at the University of New Mexico successfully detected the named attack, thus providing striking and persuasive evidence that computational immunology catches novel attacks.

**CIDF promotion.** We also conducted a successful effort to interest commercial intrusion detector vendors in standards for intrusion detection. These vendors are now heavily involved in the current IETF IDwg standards effort.

**Suggestions for further research.** Computational Immunology is much more than exercising the sliding window algorithm on sequences of program calls. While Forrest's insight into a way of characterizing program behavior is profound, there are many other areas in which to apply lessons from the vertebrate immune system to computer systems. These include the areas of

- applying anomaly-based detection to new computational entities,
- organizing system- or network-wide resistance to attacks, and
- remembering the attack and being prepared for repeats (this is what we call immunity).

One particular area of research that deserves greater attention is the problem of obtaining coverage—of ensuring that enough training data has been collected to provide a complete self database for a given application. This problem is exacerbated by changes in “normal” behavior, either because of new releases or because of gradual changes in user behavior over time. As computational immunology is used in larger and more complex systems,

the problem of obtaining coverage will become acute. Work needs to be done on automating this area.

## 8 References

1. Baker, J., F. Fung, and C. Marceau, *CORBA Immune System Installation Guide*, Odyssey Research Associates Technical Report 1998.
2. Baker, S., *CORBA Distributed Objects Using Orbix*, 1997: ACM Press.
3. Dasgupta, D. and S. Forrest, "Artificial Immune Systems in Industrial Applications," in Proceedings of the International conference on Intelligent Processing and Manufacturing Material (IPMM), 1999, Honolulu, HI.
4. Forrest, S., S.A. Hofmeyr, and A. Somajayi, "Computer Immunology," in *Communications of the ACM* **40**:10 (1997), pp. 88-96.
5. Forrest, S., S.A. Hofmeyr, and A. Somajayi, "A Sense of Self for UNIX Processes," in Proceedings of 1996 IEEE Symposium on Computer Security and Privacy, 1996: IEEE Press.
6. Ghosh, A.K., A. Schwartzbard, and M. Schatz, "Learning Program Behavior Profiles for Intrusion Detection," in Proceedings of 1st USENIX Workshop on Intrusion Detection and Network Monitoring, 1999, Santa Clara, CA.
7. Hofmeyr, S. and S. Forrest, "Immunity by Design: An Artificial Immune System," in Proceedings of Genetic and Evolutionary Computation Conference (GECCO), 1999.
8. Hofmeyr, S., S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," in *Journal of Computer Security* **6**(1998), pp. 151-180.
9. Iona Technologies, *Orbix Programmer's Guide*, 1997.
10. LPA Software, *LPA Vision User Manual 1.0*, 1997.
11. Marceau, C., *et al.*, *Architecture of a CORBA Immune System*, Odyssey Research Associates Technical Report TM-98-0005, 1998.
12. Object Management Group, *The Common Object Request Broker: Architecture and Specification (CORBA™)*, 1998: Object Management Group.
13. Object Management Group, *CORBA Services: Common Object Services Specification*, 1997.

14. Staniford-Chen, S., <http://seclab.cs.ucdavis.edu/cidf/>.
15. Stillerman, M., C. Marceau, and M. Stillman, "Intrusion Detection for Distributed Applications," in *Communications of the ACM* **42**:7 (1999).
16. Warrender, C., S. Forrest, and B. Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," in Proceedings of 1999 IEEE Symposium on Security and Privacy, 1999.

## **Appendix**

### **Intrusion Detection standards interviews**

The appendix presents the responses to a survey of intrusion detection software vendors ascertaining their interest in participation in development of standards for intrusion detection software. The companies surveyed were Axent Technologies, Centrax, Hewlett Packard, Network Associates, and SAIC Inc. The answers were elicited during face-to-face visits. The full text of the responses is included.

#### **Axent Technologies**

0. What is the name, address, Web page address and CIDF meeting attendees for this company?

Name:	Axent Technologies
Address:	2400 Research Blvd., Suite 200 Rockville, MD 20850
Web address:	<a href="http://www.axent.com">http://www.axent.com</a>
Meeting attendees:	Mark Shinbrood, Senior VP, Business Development
Meeting date:	8/5/98

1. What is the name of your product(s)?

Intruder Alert, Enterprise Security Manager, NetRecon

2. What is your framework and what platforms do you currently support?

The framework is proprietary. Our products support 55 platforms, including NT, NetWare, Solaris, AIX, HP/UX, etc.

3. What is the data format on the wire? Is it documented? Is it proprietary?

Data transfers in encrypted ASCII format

4. What type of messages do you exchange (e.g., SNMP uses the GET, SET, REPLY, and TRAP message types)?

SNMP

5. What do you build on (directly on IP, on UDP, TCP, RPC, CORBA)?

TCP, IPX

6. What do you use for security, such as secure communication?

We use our own

7. Do you provide an API to your communication infrastructure or to your IDS?

Yes

8. How do you configure your system?

We call our configuration “Drop and Detect” because we provide out of the box over 100 pre formatted policies to detect intrusions.

9. Do you see any benefit to interoperability with network management tools or standards?

Yes. Our products are currently the only products that are seamlessly integrated with Tivoli, HP-OpenView, and BMC.

Are you compliant with SNMP?

Yes.

10. What are your requirements for an intrusion detection standard?

We expect a standard to be host based and scalable to the enterprise which means that it must have a distributed architecture with console management as a key piece of the product definition.

11. Would you see an IDS standard as a benefit to your business?

Yes, we believe it would be beneficial to our customers.

If so, how?

In fact, AXENT is one of the start-up participants involved in the ICSA-sponsored Intrusion Detection Consortium. Our primary mission is to centralize terminology and common practices into some sort of unified understanding, so that the market may better understand what "Intrusion Detection" means--from the leading vendors' PoV. This give all of the vendors more credibility, and accountability in needing to cut through the marketing information, and helping the market really understand the concept of "truth in advertising."

12. Do you have trademarks, patents, or licensing fees for your communication framework?

Yes.

### **Centrax**

0. What is the name, address, Web address and CIDF meeting attendees for this company?

Name: Centrax

Address: 6540 Lusk Boulevard, Suite C-212  
San Diego, CA 92121

Phone: (619)546-2400



Web address: <http://www.centraxcorp.com>

Attendees: Paul Proctor and Chris Byrne

Date: 8/6/98

1. What is the name of your product(s)?

Detection and Response Software (eNTrax) and Centrax Audit Strategy Tool (CAST)

2. What is your framework and what platforms do you currently support?

Our platform is NT for the central console. The agents can be NT, Solaris and HP/UX AIX.

3. What is the data format on the wire? Is it documented? Is it proprietary?

The data format on the wire is proprietary and not documented for customers.

4. What type of messages do you exchange (e.g., SNMP uses the GET, SET, REPLY, and TRAP message types)?

We use SNMP for talking.

5. What do you build on (directly on IP, on UDP, TCP, RPC, CORBA)?

TCP/IP.

6. What do you use for security, such as secure communication? Do you use your own cryptographic techniques to provide for secure communication or do you assume another piece of equipment will provide it? (e.g., at one point, WheelGroup was letting the BorderGuard routers

create a virtual private network (VPN), or "encrypted sleeve" as they used to call it, provides much of the security).

Single DES.

7. Do you provide an API to your communication infrastructure or to your IDS? From C, C++, Java, Perl?

Currently, we don't provide any APIs. We have plans to provide some APIs to open up information in the database. The customers are asking for access to the information in the database. We are working on giving them that access.

8. How do you configure your system?

Configuration is defined using a GUI to specify security policies. There are audit policies and collection policies that can be set by the tool. We believe that it would be very difficult to standardize on configuration.

9. Do you see any benefit to interoperability with network management tools or standards? Are you compliant with SNMP?

We forward alerts through SNMP. HP OpenView sends messages to our system.

10. What are your requirements for an intrusion detection standard?

We would like to see standards for signatures of attacks. If we had such a standard, then records could be processed and put into a format for the product's analysis engines. All products would handle the analysis differently.

11. Would you see an IDS standard as a benefit to your business? If so, how?

We believe that commercial standards benefit the bottom line. They expand markets and open access to other resources. You should be talking with user groups as well as the commercial vendors.

12. Do you have trademarks, patents, or licensing fees for your communication framework?

Yes, we have trademarks and patents but we wouldn't suggest any required technology for the standard that would allow us to charge a fee or license technology.

13. Anything else that you would like to add?

Paul Proctor: We would like to know what DARPAs commitment is to this standard's effort. We are concerned about spending Centrax's valuable time and effort only to find that the standard is dropped due to lack of follow through.

Paul Proctor has agreed to attend the next CIDF meeting on Aug. 23 in Chicago. We welcome his participation.

### **Hewlett Packard**

0. What is the name, address, Web page address and CIDF meeting attendees for this company?

Name: Hewlett-Packard Company  
Address: 19111 Pruneridge Avenue, MS 47U12  
Cupertino, CA 95014  
Web address: <http://www.nai.com>  
Meeting attendees:

Polly Siegel, Ph.D., R&D Manager, OpenView Security  
Management Products

Mark Crosbie, Security & Intrusion Detection

Dipankar Gupta, Architect, OpenView Security  
Management Products

Rosemarie Shepley, Software Development Engineer

Meeting date: 10/5/98

1. What is the name of your product(s)?

HP\_Openview Node Sentry

2. What is your framework and what platforms do you currently support?

The term “framework” has a very specific meaning in the management software industry. HP OpenView uses a “building block” approach rather than a “framework” approach. This means that OpenView products are modular and can be implemented standalone.

Our approach is a management platform through which we will support the “plugging-in” of a variety of intrusion detection systems. We have recently announced a strategic alliance with CISCO. We support HP-UX, NT and Solaris platforms.

3. What is the data format on the wire? Is it documented? Is it proprietary?

CISCO proprietary.

4. What type of messages do you exchange (e.g., SNMP uses the GET, SET, REPLY, and TRAP message types)?

Cisco-proprietary messages. Record formats are defined in the Cisco NetRanger User's Guide.

5. What do you build on (directly on IP, on UDP, TCP, RPC, CORBA)?

Cisco-proprietary, connection-base UDP protocol.

6. What do you use for security, such as secure communication? Do you use your own cryptographic techniques to provide for secure communication or do you assume another piece of equipment will provide it? (e.g., at one point, WheelGroup was letting the BorderGuard routers create a virtual private network (VPN), or "encrypted sleeve" as they used to call it, provides much of the security).

We are exploring SNMP V.3 and SSL and IPSec. We need secure communications.

7. Do you provide an API to your communication infrastructure or to your IDS? From C, C++, Java, Perl?

N/A. Our requirement is to provide APIs to the management functions.

8. How do you configure your system?

Management and configuration is our key selling point. The HP Openview Security Management console will be used to manage a wide variety of intrusion detection systems.

9. Do you see any benefit to interoperability with network management tools or standards? Are you compliant with SNMP?

We do see benefits to this. We are compliant with SNMP.

10.What are your requirements for an intrusion detection standard?

Cisco is focusing exclusively on data exchange formats.

We need standards for configuration management and installation. In addition, standards are necessary to distribute signatures of attacks and deliver them securely. We need to perform enterprise level deployment.

11.Would you see an IDS standard as a benefit to your business? If so, how?

Yes, standards will allow us to become the management platform of choice.

12.Do you have trademarks, patents, or licensing fees for your communication framework?

Cisco does not have any trademarks, patents, or licensing fees associated with its communication framework at this time.

13.Is there anything else that you want to add?

As a company, HP is committed to standards. We are interested in becoming involved in this effort. Attending the IETF meeting and presenting our requirements is something we want to do. IDS standards are an important part of our vision for HP\_Openview Node Sentry.

## **Network Associates**

0. What is the name, address, Web page address and CIDF meeting attendees for this company?

Name: Network Associates, Inc.  
Address: 3965 Freedom Circle  
Santa Clara, CA 95054  
Web address: <http://www.nai.com>  
Meeting attendees: Michael Jones, Senior Product Manager  
Burnham H. Greeley, Development Manager  
Tom Clare, Senior Product Manager  
Aaron Bawcom, Software Engineer  
Rich Feiertag, Manager of Security  
Architecture and Methodology  
Meeting date: 10/5/98

1. What is the name of your product(s)?

CyberCop Network, CyberCop Server, CyberCop Scanner

2. What is your framework and what platforms do you currently support?

Scanning, IDS with console/sensor and server agents. Support NT and Solaris platforms.

3. What is the data format on the wire? Is it documented? Is it proprietary?

Proprietary.

4. What type of messages do you exchange (e.g., SNMP uses the GET, SET, REPLY, and TRAP message types)?

Proprietary messages. For alerts we can generate SNMP traps.

5. What do you build on (directly on IP, on UDP, TCP, RPC, CORBA)?

We build on UDP.

6. What do you use for security, such as secure communication? Do you use your own cryptographic techniques to provide for secure communication or do you assume another piece of equipment will provide it? (e.g., at one point, WheelGroup was letting the BorderGuard routers create a virtual private network (VPN), or "encrypted sleeve" as they used to call it, provides much of the security).

PGP, MS-CAPI (RSA) and proprietary security services.

7. Do you provide an API to your communication infrastructure or to your IDS? From C, C++, Java, Perl?

No.

8. How do you configure your system?

GUI to configure products.

9. Do you see any benefit to interoperability with network management tools or standards? Are you compliant with SNMP?

We do see benefits to this. We are compliant with SNMP.

10. What are your requirements for an intrusion detection standard?



We consider pre and post processing when discussing standards. By pre, we mean parameters or data that is sent **before** any kind of analysis is performed. For example, parameters or directives can be sent to sensors to direct it to report all indications of a port scan. A second example is signatures of attacks, specifically, a standard format for signatures of attack. By post, we mean everything else, such as audit data, analysis data, reports, etc. Any IDS standard must allow the system to operate in real time. We are interested in standards for pre and post IDS data.

11. Would you see an IDS standard as a benefit to your business? If so, how?

Yes, both vendor and customer benefits.

12. Do you have trademarks, patents, or licensing fees for your communication framework?

IDS technology patents.

13. Is there anything else that you want to add?

NAI wants to meet with Stuart Staniford-Chen to exchange ideas on IDS standards. They want to hear the opinions of the CIDF group chair before making an assessment of IDS standards. We are concerned about the chances of success versus failure for an intrusion detection standard. This standardization effort will take a considerable amount of resources and we need to plan carefully how we spend our time and effort. Most successful efforts standardize at the lower layers allowing companies to compete at the higher (application) layers.

## **SAIC**

0. What is the name, address, Web page address and CIDF meeting attendees for this company?

Name: SAIC  
Address: 10260 Campus Point Drive  
San Diego, CA 92121  
Web address: <http://www.saic.com/it/cmds>  
Meeting attendees: David Drake, Robert Thuleen, Dan Parker, Tommie Aycock  
Meeting date: 8/5/98

1. What is the name of the product(s)?

Computer Misuse Detection System (CMDS)

2. What is the framework and what platforms do you currently support?

CMDS uses an expert system based on the CLIPS engine developed by NASA. The system is written in C and accesses an ORACLE database. Platforms supported 1) for the CMDS Manager – Solaris 2.5.1 or higher, HP/UX 10.x, DG/UX B2 Security Option 4.12 and 2) for the CMDS agents are Solaris 2.5 or higher, HP/UX 10.x DG/UX B2 Security Option 4.12, Windows NT 4.0 and trusted Solaris. Firewall agents supported are: ANS Interlock, Raptor Eagle and Cybershield.

3. What is the data format on the wire? Is it documented? Is it proprietary?

The data format on the wire is encrypted. It is not documented for CMDS customers. The data format is proprietary and it is smaller than audit data records.

4. What type of messages do you exchange (e.g., SNMP uses the GET, SET, REPLY, and TRAP message types)?

CMDS exchanges TCP/IP messages in one direction – from the CMDS agent to the CMDS manager.

5. What do you build on (directly on IP, on UDP, TCP, RPC, CORBA)?

Same as 4.

6. What do you use for security, such as secure communication? Do you use your own cryptographic techniques to provide for secure communication or do you assume another piece of equipment will provide it? (e.g., at one point, WheelGroup was letting the BorderGuard routers create a virtual private network (VPN), or "encrypted sleeve" as they used to call it, provides much of the security).

Single DES

7. Do you provide an API to your communication infrastructure or to your IDS? From C, C++, Java, Perl?

At this time, there is no API that allows users to get data from CMDS. Users are interested in this feature. However, CMDS uses S-expressions for its data format. The reason for this is that CLIPS parses S-expressions. In addition, the CMDS data format has the equivalent of CIDF SIDS. The audit records are formatted in this way to make it easier for CLIPS to parse and analyze it.

8. How do you configure your system?

Handling configuration is important. All raw data is sent from the CMDS agents to the CMDS manager to be analyzed. The

configuration file for each CMDS agent contains the address of the target CMDS manager. Raw audit records are shipped from the CMDS agent to the CMDS manager. The CMDS manager needs to be configured to know what type of information it is receiving, for example, from an NT agent or from a Solaris agent.

9. Do you see any benefit to interoperability with network management tools or standards? Are you compliant with SNMP?

When an alert is uncovered, CMDS passes an SNMP trap to Openview. There are two problems with using SNMP messages for intrusion detection standards. The first is that some routers and firewalls throw away SNMP messages to lessen their traffic load. The second is that SNMP messages are limited in length. Long GIDOs would be a problem.

10. What are your requirements for an intrusion detection standard?

There was some talk here about why the intrusion detection companies would want to cooperate on a standard at all. The obvious answers of interoperability and data sharing were brought up which could encourage users to buy products and grow the market.

David Drake discussed his vision for this standard as follows: What is really needed here is enterprise wide protection and monitoring. How can we monitor what is going on and get the components to talk with one another? We need system wide protection, that is, self-protection on the wire and self-protection on the host. We are trying to build a trusted environment on top of something that we can't trust. At an abstract level, a security management framework is the goal and should be the goal of the standard. CIDF focuses on one piece to try to achieve this objective.

11. Would you see an IDS standard as a benefit to your business? If so, how?

A benefit that we perceive is a standard that would require different IDS to have a common rule set for detecting an attack. This is analogous to a standard on safety belts for automobiles. You have different designs of safety belts, but all conform to the standard.

12. Do you have trademarks, patents, or licensing fees for your communication framework?

Yes, SAIC has trademarks and patents for CMDS. We don't charge any licensing fees for communication framework.

13. Is there anything else that you want to add?

We are checking into availability of resources to apply to a standards effort. SAIC will get back to ORA on this issue.